Optimización Energética de la Reserva de Espacios en Edificios Empresariales

Índice de contenido

1 Introducción

1.1 Objetivos

El objetivo de este proyecto es la adaptación, implementación y evaluación de una serie de algoritmos que permitan la calendarización de un conjunto de eventos entre entidades e instalaciones, en nuestro caso específico 'Reuniones' y 'Espacios', localizando soluciones factibles que cumplan las restricciones definidas y tratando de acercarse al coste óptimo, de acuerdo con el modelo energético establecido. Se quiere establecer que algoritmo produce las mejores soluciones siguiendo los criterios de coste energético, optimalidad y tiempo de cálculo para los diferentes tamaños y características de problemas a tratar.

El proyecto ha sido desarrollado en los laboratorios 4C[1] de la *University College Cork*[2], como parte del **proyecto ITOBO**[3], por lo que está orientado a reuniones empresariales entre diferentes agentes en el interior de edificios corporativos. Sin embargo, su funcionamiento y diseño permitiría adaptarlo a otros problemas de reserva de recursos con relativa facilidad. Este proyecto, por lo tanto, además de encontrarse en funcionamiento activamente en las instalaciones de algunos de nuestros socios tecnológicos, como por ejemplo la empresa de domótica Cylon Controls[4], puede entenderse como una comparativa de diferentes casos de búsqueda combinatoria informada para el rango de problemas determinados a resolver en este contexto.

Otro **objetivo** del proyecto consiste en presentar una interfaz de demostración para los diferentes congresos y presentaciones a socios tecnológicos donde se desarrolle la solución y, aunque inicialmente se implementaron los algoritmos básicos en modo únicamente textual, más adelante fueron adaptados a una solución web intuitiva en su uso y adecuada para representar los problemas, su conjunto de soluciones y facilitar la comparativa de los métodos subyacentes.

Aunque el entregable principal de este PFC es la herramienta de calendarios, sus algoritmos y su interfaz web asociada, se presentarán otros algoritmos, también comprendidos dentro del contexto de ITOBO, orientado a la predicción del confort de los usuarios analizando bases de datos de realimentación (*feedback*) real recogidas por diferentes universidades. Gracias a estos algoritmos y sus resultados, se realizaron una serie de publicaciones académicas que se comentarán en la sección 'Aportaciones', dentro del capítulo 'Conclusiones'.

Un factor a tener en cuenta y que resultó positivo durante la realización del trabajo fue la comunicación con otros desarrolladores, situando este PFC como una pieza de un proyecto de gran envergadura que debía colaborar con todos los demás módulos, por lo que el código habría de ser reusable, autoexplicativo y modular. La experiencia de estos desarrolladores, que aún mantienen el

1 http://4c.ucc.ie/web/index.jsp
2 http://www.ucc.ie/en/
3 http://zuse.ucc.ie/itobo/
4 http://www.cylon.com/

código en las sucesivas demostraciones y despliegues que se están llevando a cabo[5] ha sido bastante positiva (por ejemplo con la interfaz de confort para usuarios de ITOBO), a lo cual ha contribuido en gran medida el uso de un sistema de control de versiones (*git*). La interfaz web también está desarrollada siguiendo un patrón de Modelo-Vista-Controlador para aumentar su modularidad y reusabilidad.

1.2 Contexto del proyecto

Los principales objetivos del proyecto ITOBO son la automatización, toma de decisiones y ahorro energético en el contexto de los grandes edificios empresariales. Surge financiado por un consorcio público-privado integrado por entidades como Intel Irlanda o ERI (*Environmental Reseach Institute*). El proyecto pone de manifiesto la carencia de un sistema de control que optimice los diferentes subsistemas o procesos:

- Mantenimiento.
- Calendarización de mantenimiento y reuniones.
- Automatización del sistema HVAC (*Heat, Ventilation, Air Conditioning*).
- Control de inventario (RFID).
- Automatización del sistema de iluminación (mediante sensores y aprendizaje de las preferencias de usuarios).
- Registros, estadísticas y presentación adecuada del histórico de consumo, tanto a nivel general como por zonas.

Se engloba pues en la categoría de "Edificios Inteligentes". Pese a existir gran variedad de ofertas empresariales en este mismo campo, los estudios previos demuestran que estas soluciones son parciales (cubren muy pocos de los puntos mencionados) y rígidas: el sistema no está correctamente abstraído y modularizado, siendo necesario, como ejemplo, el uso de unos determinados sensores del fabricante que son los únicos compatibles con el software central. El sistema ITOBO se encuentra en estos momentos en fase de desarrollo y ha sido desplegado parcialmente con resultados positivos en cuanto al ahorro energético en las instalaciones de *Cylon Controls* en Irlanda o en el edificio *Messeturm* en Frankfurt[6], gestionado por HSG-Zander[7].

1.3 Alcance

La complejidad del problema de la calendarización se engloba dentro de los problemas *NP-hard*, por lo que existe una aproximación para el modelo de transferencia de energía, calor entre habitaciones en nuestro caso, del que hablaremos más adelante.

5 El proyecto ITOBO continuará hasta el 2013
6 http://es.wikipedia.org/wiki/Messeturm
7 http://www.hsgzander.com/

En todo momento había que tener en cuenta la compensación entre optimalidad de la solución ofrecida y el tiempo necesario para obtenerla. Por ello, dado que inicialmente sólo se contaba con un *solver* de "Programación con restricciones" o -*Constraint Programming*, que era el principal área de conocimiento del laboratorio donde comenzó el proyecto[8]-, un método de búsqueda completa, dos nuevos *solvers* de búsqueda local fueron añadidos para estudiar sus comparativas en tiempo y memoria. En el contexto de este proyecto hablaremos pues de tres algoritmos.

1. El algoritmo de *Constraint Programming*, abreviado como CP.
2. El algoritmo de búsqueda local de 'ascenso de colina' mediante la técnica de *Iterated Local Search,* abreviado como ILS.
3. El algoritmo de búsqueda local basada en algoritmos genéticos, abreviado como GEN.

Añadir estos dos últimos *solvers* fue una decisión muy beneficiosa, tanto para resolver algunos tipos de problemas, tal y como veremos en las comparativas, como para mejorar la modularidad del proyecto, dado que ciertas partes del programa se refactorizaron para hacerlas más reusables y poder integrar nuevos motores con el mínimo esfuerzo posible. Adicionalmente, se diseñó una jerarquía de objetos para ambas búsquedas locales que nos permitiese reutilizar la mayor parte de las entidades comunes, minimizando el código específico.

La solución debía ser además rápidamente adaptable a cambios en las restricciones y condiciones del modelo energético, dado que el módulo de mantenimiento del edificio podía generar eventos que desplazasen a las reuniones ya establecidas, o simplemente los participantes podían modificar su agenda, cancelar eventos, etc.

El proyecto ofrece diferentes herramientas para proporcionar soluciones y comprobar el funcionamiento de los *solvers*:

- Un **conjunto** de **pruebas** unitarias para asegurar el correcto funcionamiento de las entidades básicas que conforman el problema.
- Un **módulo** del programa que **comprueba** que las soluciones cumplen todas las restricciones impuestas por el problema y que el coste de la solución expresado por el *solver* coincide con el calculado de manera independiente. La implementación en una fase temprana del proyecto de este módulo fue crucial para descubrir errores en la programación y asegurar que las nuevas optimizaciones en cualquiera de los algoritmos jamás rompían ninguna de las restricciones. Este módulo recibe el problema y la solución de manera independiente, sin compartir estructuras de datos con los *solvers* para conseguir así un análisis de 'caja negra'.
- Un **módulo** encargado de **generar problemas** de diferentes dimensiones y características (se desarrollará en punto 3.7). Debido a que los problemas de gran tamaño son costosos de producir manualmente y, lo que es más importante, pueden no tener solución factible, en algunos casos, por implicaciones de las restricciones no obvias a primera vista, era

8 http://4c.ucc.ie/web/index.jsp

importante contar con una herramienta que permitiese la generación de problemas de diferentes tamaños, con diferente grado de restricción y que garantizase por lo menos una solución factible.

1.4 Trabajo previo

En el desarrollo de este proyecto se aprovechan una serie de herramientas y *frameworks* de programación existentes. Dado que el desarrollo de un motor de *Constraint programming* adecuado es una tarea que representaría una cantidad inaceptable de tiempo para el objetivo del proyecto, se usa uno de los motores desarrollados al cabo de tres años por académicos del propio laboratorio, *Choco*[9] .

Entre las herramientas más destacables en el desarrollo de este proyecto se encuentran:

- El sistema de control de versiones *git*
- El *java IDE* Eclipse para la versión inicial en modo textual
- El *java IDE Netbeans*, fue de gran utilidad en el desarrollo de la versión web, debido a su servidor *Glassfish* embebido que sincronizaba y desplegaba automáticamente los nuevos contenidos del proyecto.
- El *framework* de java *Spring*, mediante el cual se implementó el patrón de modelo-vista-controlador en la web.

Además del uso de multitud de librerías de Java y Python, para el acceso a bases de datos, procesamiento XML y cálculos numéricos. Otros lenguajes usados extensivamente han sido SQL, HTML, CSS y Javascript.

El trabajo previo consistió principalmente en el estudio del problema, sus particularidades y topología del espacio de soluciones; los diferentes algoritmos a evaluar para su resolución, como hemos mencionado anteriormente *Constraint Programming , Iterative local search y Genetic,* así como las tecnologías web necesarias para hacer una representación gráfica en una web dinámica con abundante uso de Ajax y JSON.

9 http://www.emn.fr/z-info/choco-solver/

1.5 Líneas generales

1.5.1 Estrategia

El proyecto se ha dividido en

- **Investigación:** Comprendiendo el análisis detallado del problema que abordamos, las diferentes soluciones y métodos propuestas en la literatura relacionada a problemas similares, así como implementaciones disponibles, las mecánicas de trabajo para el diseño y optimización de algoritmos *Constraint Programming.* El tiempo a emplear en esta fase se fijó en un mes.
- **Desarrollo del primer *solver:*** Con los conocimientos adquiridos en la fase de investigación se llevó a cabo el primer *solver*, presentando problemas de rendimiento en sus primeras versiones, por lo que el modelo tuvo que ser reconstruido tres veces hasta que se llegó a una implementación satisfactoria. Sus problemas de rendimiento, al ser una búsqueda completa, hicieron necesario el estudio e implementación de diferentes algoritmos con el fin de alcanzar una plataforma más completa. El tiempo a emplear en esta fase se fijó en mes y medio.
- **Desarrollo de la versión web:** Inicialmente diseñada con el único objetivo de presentar una interfaz gráfica para las demostraciones a nuestros socios, gradualmente se optó por el uso de gran variedad de tecnologías web para aumentar su modularidad y presentación. El tiempo a emplear en esta fase se fijó en diez días.
- **Desarrollo de dos algoritmos de búsqueda local:** Tras sufrir problemas de rendimiento con el primer algoritmo, se planteó la implementación de búsquedas locales para aportar mayor completitud al proyecto y, sobre todo, para poder plantear comparativas sobre el mismo problema. El tiempo a emplear en esta fase se fijó en un mes y medio.
- **Integración:** Durante esta fase se adaptaron los algoritmos y la jerarquía de clases para unificar las interfaces de entrada y salida y adaptarlas a los contextos textuales y web, se definieron unos formatos unificados para la representación del problema y se llevaron a cabo pruebas de rendimiento y tests de conformidad de la soluciones. El tiempo a emplear en esta fase se fijó en veinte días.
- **Documentación y comparativas:** Recopilando todo el material acumulado de investigación, artículos publicados y resultados se llevo a cabo la escritura de una versión unificada de la memoria, existiendo ya documentación individual de diferentes partes del proyecto, generación de los diferentes problemas para las tablas de comparativa y la ejecución de los mismos. El tiempo a emplear en esta fase se fijó en ocho días.

Id	Paquete de trabajo	Inicio	Fin
1	Investigación	T_0	T_0 + 1m
2	Desarrollo CP	T_0 + 1m	T_0 + 2,5m
3	Desarrollo web	T_0 + 2,5m	T_0 + 2,8m
4	Desarrollo LS	T_0 + 2,8m	T_0 + 4,2m
5	Integración	T_0 + 4,2m	T_0 + 4,8m
6	Documentación	T_0 + 4,8m	T_0 + 5m

Tabla 1: Cronograma estimado

Finalmente, el desarrollo de los algoritmos de búsqueda local se llevaron a cabo ligeramente antes de lo esperado, mientras que el tiempo de desarrollo de la versión web se triplicó hasta convertirse en un mes de trabajo. La documentación e integración llevaron a cabo ligeramente más tiempo del planeado.

Id	Paquete de trabajo	Inicio	Fin
1	Investigación	T_0	T_0 + 1m
2	Desarrollo CP	T_0 + 1m	T_0 + 2,5m
3	Desarrollo web	T_0 + 2,5m	T_0 + 3,5m
4	Desarrollo LS	T_0 + 3,5m	T_0 + 4,5m
5	Integración	T_0 + 4,5m	T_0 + 5,2m
6	Documentación	T_0 + 5,2m	T_0 + 5,5m

Tabla 2: Cronograma real

1.5.2 Métodos y Técnicas

La metodología principal de desarrollo que se ha seguido obedece al **método de prototipo evolutivo más integración continua**, ya que debido a las frecuentes presentaciones y congresos se mostraba parte de la funcionalidad en una etapa temprana. Adicionalmente, muchas de las pruebas de rendimiento para nuevos modelos resultaban fallidas, por lo que era indispensable conocer y salvaguardar los puntos “seguros” del desarrollo. El prototipo propuesto iba adaptando progresivamente las diferentes restricciones del problema y mejorando sus tiempos de respuesta.

El prototipo evolutivo y la integración continua son técnicas recomendadas cuando se requiere mostrar funcionalidad frecuentemente, se requiere adaptación temprana a cambios en los requisitos y se necesitan datos empíricos para comprobar la validez de las sucesivas evoluciones, dado que es muy difícil o incluso imposible tener un dato fiable sobre el rendimiento en fase de

diseño, tal como era el caso.

Para elaborar este modelo fue crucial el uso de dos herramientas: un sistema de **control de versiones** que permitía estudiar y comentar los cambios que habían producido cada mejora, así como volver a versiones anteriores tras un intento fallido y, en segundo lugar, el módulo de **comprobación de soluciones**, más la batería de problemas de entrada (de diferentes tamaños y también conteniendo casos no factibles), que facilitaba el descubrimiento de errores en la implementación o incluso en el diseño del algoritmo.

Esta metodología permitió el rápido desarrollo de propuestas de diseño y cambios en los modelos de resolución, tomando decisiones ágiles sobre la conveniencia de continuar con la integración de un modelo particular y adaptando las opiniones recogidas en las diferentes demostraciones.

1.5.3 Herramientas

Desarrollo

Durante el desarrollo de este proyecto se han utilizado principalmente las siguientes herramientas:

- *Eclipse IDE*: Entorno integrado de desarrollo, principalmente para Java, aunque acepta *plug-ins* para otros lenguajes. Fue de gran utilidad gracias a la versatilidad de su interfaz, pero sobre todo, gracias a su *framework* de *debugging*.
- *Netbeans IDE*: Entorno integrado de desarrollo, usado principalmente por su buena integración con los servidores de aplicaciones java en la web.
- *Git*: Herramienta de control de versiones distribuida, rápida, de fácil manejo y gran potencia. Se ha usado también su versión gráfica '*gitk*' para visualizar los cambios entre versiones.
- *Firebug*: Extensión de *firefox* que permite el *debugging* de los programas en javascript/Ajax necesarios para la implementación de la versión web .

Durante las distintas tareas de desarrollo también han sido de gran utilidad otras muchas herramientas básicas del sistema *GNU/Linux*, como el editor *vim* o la herramienta de copias de seguridad *rsync*.

Documentación

Las herramientas utilizadas para generar esta documentación han sido:

- *Libreoffice*: Entorno ofimático libre y de propósito general.
- *Dia*: Herramienta ligera de generación de diagramas UML.

- *Inkscape*: Herramienta libre de dibujo vectorial.

1.6 Componentes adicionales

Además del módulo de calendarización de reuniones, expondremos brevemente otros módulos que forman parte del proyecto ITOBO que fueron elaborados durante mi estancia en los laboratorios 4C de la Universidad de Cork:

1.6.1 PMV prediction tool

Desarrollada en el lenguaje Python, se basa en las mediciones PMV (*Predicted Mean Vote*) de confort realizadas en diferentes zonas climáticas por la Universidad de Macquarie y disponible en bases de datos para su uso en otros estudios[10]. Empleando diversas técnicas estadísticas se categoriza a los usuarios del edificio por oficinas y se calcula la distribución de satisfacción dando diferentes pesos a los datos históricos dependiendo de si provienen del mismo usuario, de la misma oficina o de una zona climática similar. Los algoritmos resultantes produjeron valores más aproximados a la reacción real de los usuarios que los disponibles en ese momento en la literatura del campo, dando lugar a la publicación de dos artículos académicos.

1.6.2 User Feedback Interface

Esta interfaz de usuario tiene el objetivo de recoger los datos de campo necesarios para los distintos algoritmos de toma de decisiones. Gracias a ella, los usuarios pueden mostrar su opinión respecto a los valores de temperatura, humedad y luminosidad de su puesto de trabajo de acuerdo a la escala *ASHRAE PMV*. Esta interfaz obtiene los datos del usuario y de los diferentes sensores próximos a él de la base de datos centralizada, *Oracle* en nuestro caso, y está conectada a los algoritmos de decisión de los actuadores, gracias a lo cual pudimos llevar a cabo demostraciones en las que se activaban elementos del HVAC en tiempo real de acuerdo a la media de *feedback* de los usuarios. Mediante esta interfaz es también posible hacer peticiones de mantenimiento y conectar a la interfaz de calendarización de reuniones.

Ambos módulos serán descritos en más profundidad en los anexos.

1.7 Estructura de este documento

La documentación del proyecto consta de la memoria principal y una serie de anexos, acompañados de la bibliografía y un glosario de términos y acrónimos.

10 http://aws.mq.edu.au/rp-884/ashrae_rp884.html

La memoria está organizada en seis capítulos: Introducción, Definición detallada del problema, Trabajo desarrollado, Evaluación, Conclusiones y Anexos.

- En el **primer** capítulo se describe el objetivo de este proyecto, su alcance, trabajo previo, el contexto del proyecto ITOBO donde se encuadra, las líneas generales de desarrollo y un comentario de los módulos adicionales generados en este contexto.
- En el **segundo** capítulo define con la mayor brevedad posible el problema a abarcar, sus componentes, sus restricciones y el criterio de optimalidad.
- El el **tercer** capítulo desarrollan las soluciones adoptadas, su diseño, características y los criterios con los que se llevaron a cabo las implementaciones, además de proporcionar explicaciones de los fragmentos de código más relevantes.
- En el **cuarto** capítulo expone una comparativa de los algoritmos implementados basada en diferentes criterios, como calidad de la solución y rendimientos en tiempos, acompañada de gráficas explicativas y conclusiones.
- En el **quinto** capítulo se presenta, basándose en los datos anteriores, las **conclusiones** sobre los algoritmos implementados, así como los principales aciertos y errores durante el desarrollo del proyecto y experiencia adquirida a través de ellos.

Los anexos incluidos en el **sexto** capítulo son los siguientes:

- Anexo A - Manual de usuario de la web: a través de captura de pantallas y explicaciones, una vez conocido el contexto del problema, podremos usar la versión web del mismo para comprobar de forma conveniente y gráfica las diferentes soluciones que nos brindan los algoritmos. Como veremos, es posible salvar y editar un problema en todo momento para comprobar el comportamiento ante cambios en las condiciones del mismo.
- Anexo B - *PMV prediction tool*: Se desarrollará brevemente este módulo del proyecto ITOBO, las convenciones estadísticas usadas y su funcionamiento. También se comentarán las bases de datos usadas como entrada y sus resultados. Como subsección de este algoritmo encontraremos dos publicaciones relacionadas con sus resultados "*Learning User Preferences to Maximise Occupant Comfort in Office Buildings* " y *"Predicting the desired thermal comfort conditions for shared offices*", que serán descritos brevemente en los puntos 4.2.1.1 y 4.2.1.2 e incluidos íntegramente en la versión digital.
- Anexo C - *User feedback interface*: Se presentará la mencionada web para recoger el *feedback* de los usuarios y se comentarán sus implicaciones en el sistema central de HVAC y futuras extensiones que serán llevadas a cabo en el contexto del proyecto ITOBO.

2 Definición detallada del problema

A continuación se detalla el problema que debemos abarcar en este proyecto, siguiendo los criterios de factibilidad (el resultado no debe violar ninguna de las restricciones) y optimalidad de acuerdo al modelo energético que nos dicta cuál es el coste energético de la solución alcanzada.

2.1 Componentes

Calendario: Define el intervalo de tiempo que vamos a tratar en el problema, sobre el que se llevarán a cabo las reuniones y que establece cuales son las "ranuras" de tiempo a considerar por los algoritmos. Básicamente es una estructura de datos bidimensional, en la versión web es posible asignar etiquetas a las columnas y filas arbitrariamente (p.ej. 'días' y 'horas' o 'meses' y 'días'), lo cual no tiene ninguna repercusión real en los algoritmos de resolución. El calendario define, en resumen, el dominio de las variables (reuniones) a las que debemos asignar un valor. Ver Figuras 18 y 22 presentes en el 'Manual de usuario de la web' Anexo A.

Agente: Cada una de las entidades disponibles para los eventos, puede considerarse una persona, una empresa o cualquier otra entidad. Cada Agente tiene asociado su calendario particular, del mismo tamaño que el calendario general, pero con información de las ranuras donde este agente estará disponible para el evento.

Habitación: Una habitación representa el lugar donde se llevará a cabo el evento. Cuenta con un calendario particular del mismo modo que el agente, dado que puede tener previstos eventos y acciones de mantenimiento que inhabilitan la sala temporalmente. Adicionalmente tiene un máximo de ocupación (número de agentes máximo que pueden acudir a un evento), y objetos contenidos en la sala que pueden ser necesarios para los eventos -proyectores, conectividad inalámbrica, mesa de reuniones, etc -.

Reuniones: Componen, a bajo nivel, las variables que tendremos que organizar en nuestro calendario. Una reunión está compuesta por una lista de agentes interesados, tiene una duración en ranuras de tiempo definida y necesita un conjunto de items u objetos.

Las entidades "Relación Energética" entre habitaciones y las características energéticas de las mismas serán definidas más abajo en el modelo energético.

2.2 Restricciones

El problema presenta las siguientes restricciones, que en gran parte se derivan de manera natural de los componentes mencionados:

- Agentes: un agente no puede ser asignado en una ranura que tenga marcada como no disponible. Un agente no puede estar en dos reuniones al mismo tiempo
- Habitaciones: Una habitación no puede contener más agentes de los indicados en su perfil, no se puede asignar una ranura que tenga marcada como no disponible, no se puede asignar una reunión a una habitación si esta no contiene alguno de los elementos requeridos por el mismo.
- Reuniones: Una reunión no se puede solapar con ninguna otra en la misma habitación, no se puede acortar el tiempo requerido por una reunión ni partirla entre varios días diferentes.

2.3 Modelo energético

Una vez definidos los criterios de factibilidad de nuestras soluciones, nuestro objetivo secundario es hallar la solución más eficiente desde el punto de vista energético que cumpla con todas estas restricciones.

Pese a existir herramientas de simulación de la transferencia de calor entre habitaciones en el propio contexto del proyecto ITOBO, la cantidad de soluciones factibles presentes en un problema de tamaño pequeño – medio (decenas de miles de acuerdo al algoritmo CP) hacen totalmente impracticable un cálculo tan detallado, por lo que desde un primer momento se diseña un modelo simplificado de compromiso que nos permite calcular de forma rápida el coste energético de la solución encontrada.

En nuestro modelo las habitaciones tienen dos características energéticas, el **coste de calentar la habitación a una temperatura adecuada previamente a un evento** -recordemos que en un contexto de edificios inteligentes, el sistema está informado de la ocupación de la salas y por lo tanto solo aporta calor cuando sea necesario- y el coste de mantener esta habitación a la temperatura adecuada durante una ranura de tiempo.

Tengamos en cuenta que si la habitación ha sido usada en la ranura temporal anterior, no tendremos que "pagar" el coste inicial, dado que se mantiene caliente. Otra consideración importante del modelo energético es **la transferencia de calor entre habitaciones**. Como la experiencia y los datos de campo nos indican, será necesario menos energía para mantener caliente una habitación si las habitaciones adyacentes se encuentran en funcionamiento en el mismo momento. Es por ello que existe un quinto componente en el problema: las relaciones entre habitaciones, expresadas en cantidad de energía ahorrada por la habitación si una determinada habitación adyacente se encuentra en funcionamiento en ese mismo momento.

Así pues, en este documento hablaremos de la *StartupEnergy* o energía necesaria para calentar inicialmente una habitación y la *OperatingEnergy* o energía necesaria para mantener el calor durante una de sus ranuras de tiempo.

Partiendo de esta definición se busca una solución factible y lo más óptima posible para, posteriormente, ser enviada al módulo de comprobación de soluciones, el cual verificará que todas las restricciones se cumplen y que, efectivamente, la energía consumida es exactamente la que especifica el algoritmo de acuerdo al modelo.

La entrada y salida de los algoritmos se realiza mediante documentos XML, con una semántica adaptada al caso. El módulo de *parsing* utiliza en primer lugar un documento en lenguaje XML Schema (xsd) para validar el formato y posteriormente localiza contradicciones relacionadas con el ámbito del problema. Si la entrada es válida, devuelve las estructuras de datos pertinentes con la representación del problema. Se prefirió el formato XSD sobre DTD por su mayor precisión en la declaración de entradas válidas y por su orientación más programática.

3 Trabajo desarrollado

3.1 Diseño de la jerarquía de clases

El diseño del sistema ha sido orientado a objetos, usando como lenguaje principal Java. Desde un primer momento y debido a los requisitos que se indican en la sección “Métodos y técnicas” de la introducción, se intentó llevar a cabo un buen diseño de la jerarquía de clases para reutilizar al máximo el código disponible, evitar repeticiones y presentar mayor modularidad. El proyecto se estructuró en cuatro paquetes básicos: el modelo, que contenía las entidades mencionadas, así como las operaciones sobre las mismas, el paquete de búsqueda CP, el paquete de búsqueda local, que contenía los paquetes para ILS y GEN, agrupando la mayor cantidad de funcionalidad posible en el paquete “padre” para poder implementar nuevos algoritmos de búsqueda local fácilmente, reusando elementos y el paquete 'útil' que contenía el módulo de comprobación, así como utilidades para gestionar los documentos XML y HTML de entrada y salida.

El diseño del modelo en un paquete separado y encapsulado facilitó la posterior implantación del patrón modelo-vista-controlador en la web, siendo portado sin apenas cambios. Se detallará únicamente la versión en modo texto del programa, dado que la versión web tiene muchas más clases, implementando los controladores no relevantes para el funcionamiento básico de los *solvers* y que serán desarrolladas en la sección 'Desarrollo Web'

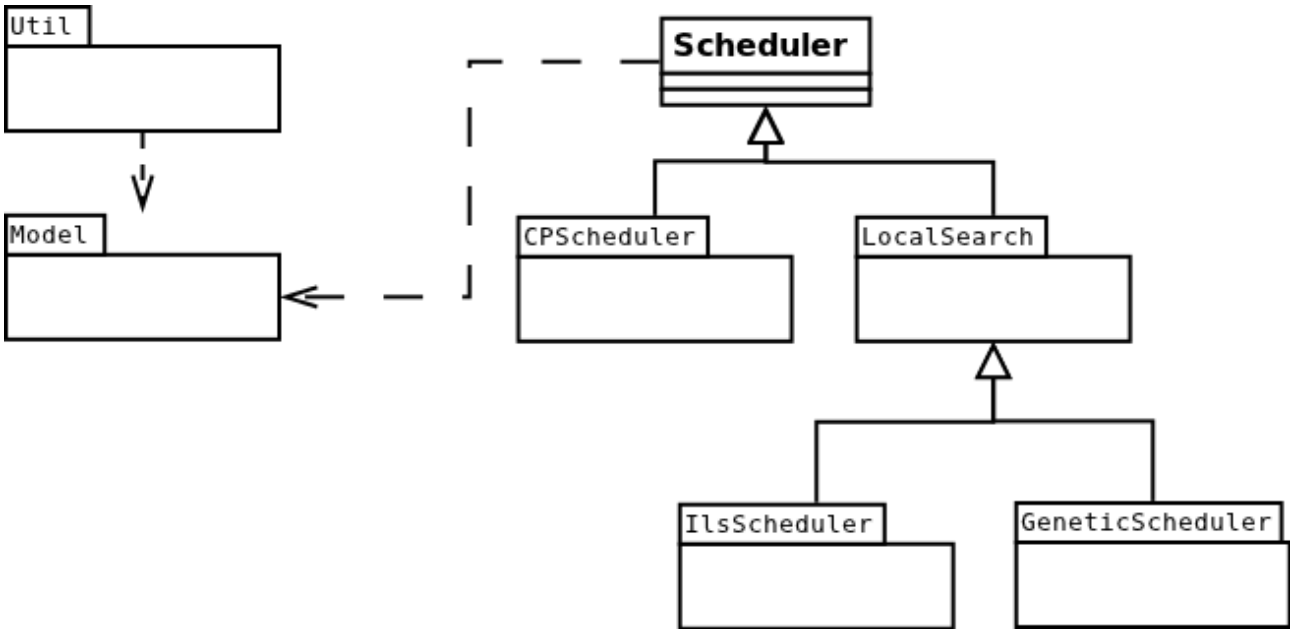

Figura 1 – Diagrama general de paquetes software

Los paquetes CPScheduler y LocalSearch se consideran una implementación de la clase Scheduler, que contiene las estructuras básicas del problema así como los métodos de entrada

salida. Se considera que la clase scheduler y el paquete de utilidades Util dependen del modelo por que cualquier cambio en el modelo, implicaría la reescritura parcial de ambos.

A continuación desarrollaremos brevemente los paquetes más relevantes

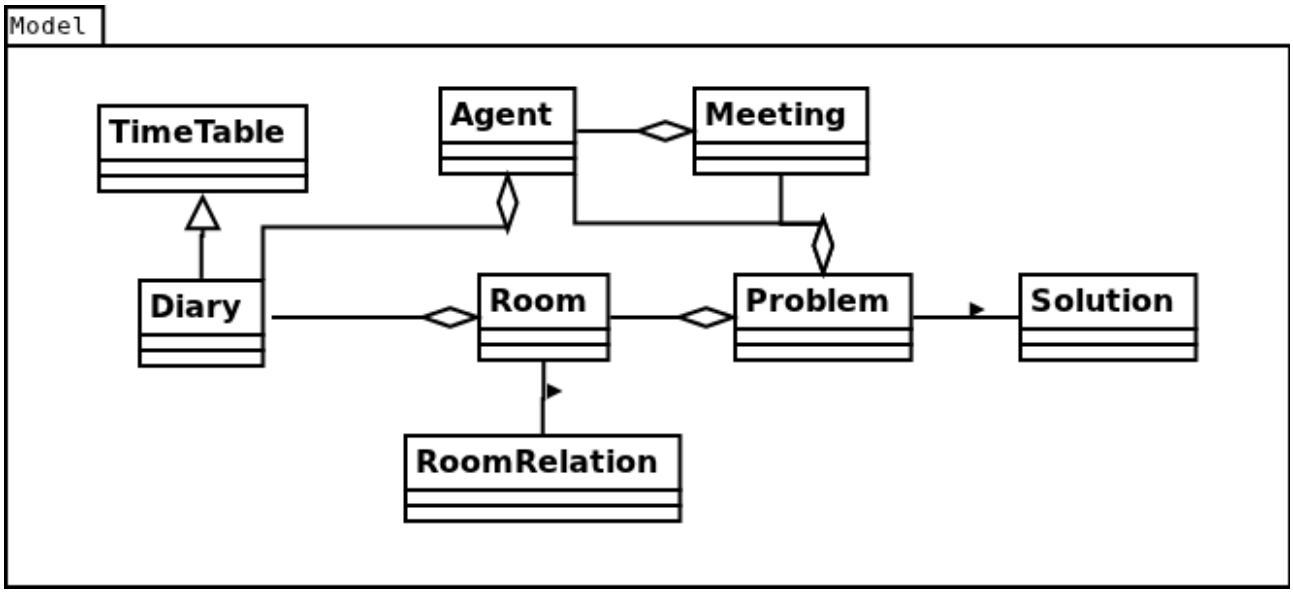

Figura 2 – Diagrama del paquete 'Model'

Como podemos ver, la clase central del paquete es el Problema. Un problema está compuesto por *n* Agentes, Habitaciones y Reuniones. La clase 'TimeTable' se correspondería con el componente 'Calendario', desarrollado en la descripción del problema, mientras que los diarios son los calendarios marcados con la disponibilidad de Agentes y Habitaciones. Las '*RoomRelation*' relacionan la transferencia de energía entre las diferentes instancias de Habitaciones. Cada problema tiene asociado un numero *n* de soluciones.

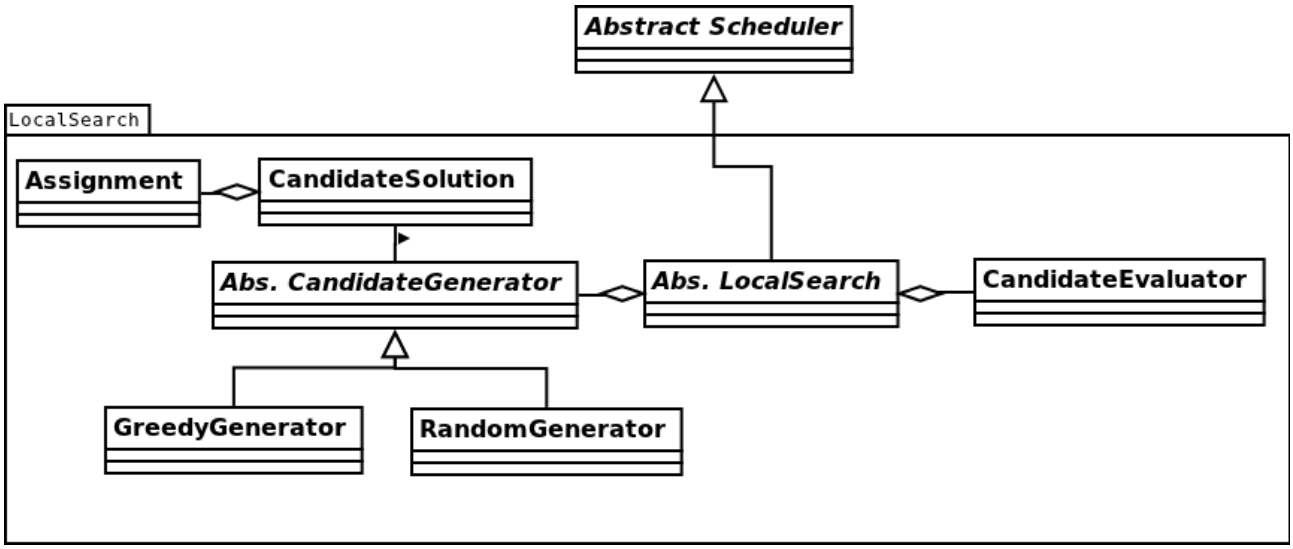

Figura 3 – Diagrama del paquete 'LocalSearch'

En este paquete son de especial relevancia las clases '*CandidateGenerator*' y '*CandidateEvaluator*'. Como su nombre indica '*CandidateGenerator*' es la encargada de generar nuevos 'especímenes' a considerar en la búsqueda local. Esta clase es abstracta, siendo extendida por dos clases, el '*GreedyGenerator*' y el '*RandomGenerator*', que serán detalladas más adelante. Como podemos ver en el diagrama, el '*CandidateGenerator*' generará candidatos a soluciones, compuestos de asignaciones a las variables. El '*CandidateEvaluator*', como su propio nombre indica, evaluará el coste de la solución candidata.

Es importante destacar que no podemos desechar directamente las soluciones no factibles (que violen alguna de las restricciones) en los métodos de búsqueda local, ya que el objetivo es irnos aproximando progresivamente a una solución primero factible y, posteriormente, óptima. Es por ello que a partir del modelo se calcula el máximo teórico de coste para este problema (todas las reuniones en la habitación más cara y sin ningún ahorro de energía por reuniones adyacentes) y por cada violación de una regla se añade al coste el [máximo teórico + 1]. De esta forma, cualquier solución con *n-1* violaciones de las restricciones **siempre** será más barata que una solución con *n* violaciones. Asimismo, una solución factible siempre será más barata que una solución que contenga una violación. Como es natural, dentro de las soluciones factibles, será más barata la más óptima energéticamente.

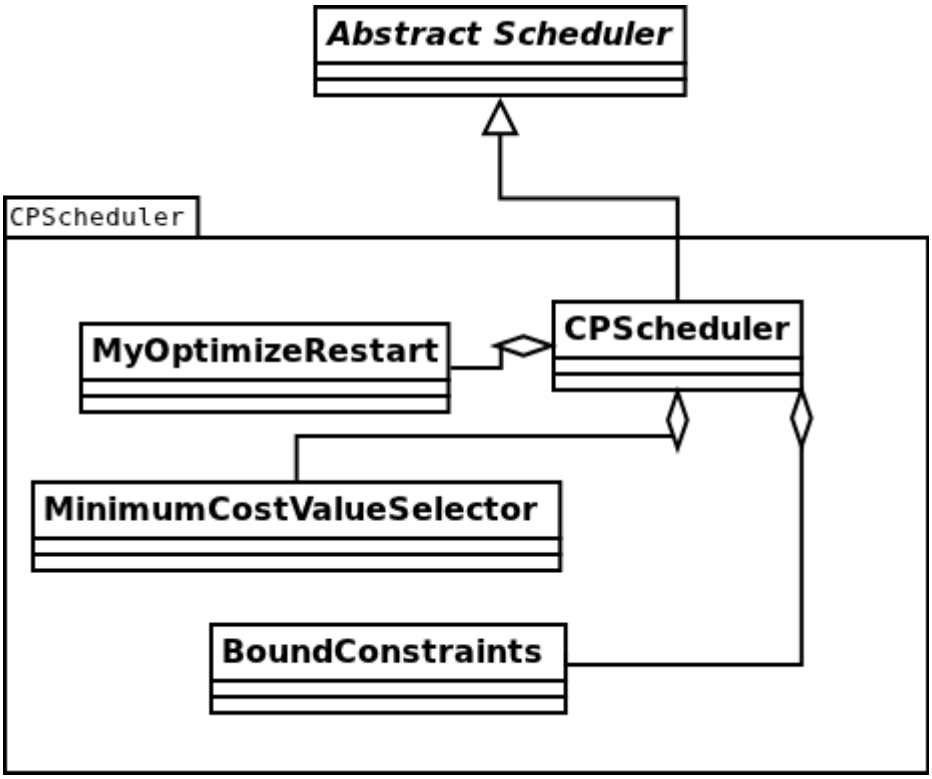

Figura 4 – Diagrama del paquete 'CPScheduler'

Podemos observar que la clase *CPScheduler* es una implementación de la clase abstracta '*Scheduler*'. Las clases '*MyCPSolver*', '*MinimumCostValueSelector*' y '*MyOptimizeRestart*' extienden directamente de clases del paquete de resolución de *Constraint Programming Choco*, mencionado anteriormente. Gracias a estas tres clases podemos ajustar nuestra búsqueda y límites de coste superior e inferior a nuestro dominio específico. Todo este paquete será desarrollado en más detalle en la sección correspondiente.

3.2 Algoritmo Constraint Programming

3.2.1 Introducción

En primer lugar vamos a definir brevemente el contexto de este paradigma de programación *Constraint Programming* o programación orientada a restricciones. La CP es una forma de programación declarativa, ya que no especifica explícitamente una serie de pasos a tomar sino más bien las condiciones que debe satisfacer una solución válida. [11] [12] [13]

11 http://kti.mff.cuni.cz/~bartak/constraints/
12 http://www-module.cs.york.ac.uk/cop/cso0.pdf
13 http://homepages.laas.fr/lopez/cours/CP/SEM-CSP.pdf (Ver Bibliografía en la última página)

El problema está definido a partir de la declaración de variables y sus dominios y la declaración de restricciones entre éstas. Así, como ejemplo, podemos establecer que un conjunto de variables deben tener valores diferentes entre ellas o que la variable *X* tiene que ser por lo menos *Y+10*.

La CP es generalmente un algoritmo de búsqueda completa, donde se genera un árbol de búsqueda en base a las variables declaradas y sus dominios. Un concepto básico de la CP es la propagación de restricciones, eliminación de valores del dominio de las variables que se deducen de las restricciones.

A modo de ejemplo simple supongamos la variable *X* con dominio entero *[1..10]* y la variable *Y*, con el mismo dominio. Si hemos declarado la restricción *X >= Y+5*, el motor de CP puede eliminar en la propagación previa los valores *[1..5]* de la variable *X,* así como los valores *[6..10]* de la variable *Y*. Durante la búsqueda también se lleva a cabo esta propagación, por lo que si el valor de *Y* que hemos seleccionado en ese momento de la búsqueda es '4', restringiremos el dominio de *X* a *[9,10]*. Aplicando propagación previa y dinámica, hemos reducido el número de nodos del árbol de búsqueda de 100 a 15. La propagación de dominios se transmite entre todas las variables posibles, de tal forma que si *X* tiene una restricción sobre *Y*, que tiene una restricción sobre *Z*, la eliminación de un valor del dominio de *Y* por la restricción *Y→ X*, puede provocar la eliminación de valores en *Z*.

Muchos entornos de programación de CP, además de proveer satisfacción de restricciones, suministran el motor para asignar un 'coste' a una variable o conjunto de variables dependiendo del valor seleccionado, por lo que son capaces de buscar no solo una solución factible, sino lo más óptima posible. En estos casos es esencial definir un límite superior e inferior para el coste de nuestro problema.

La definición de un límite inferior es importante ya que nos permitirá hacer poda de dominios y nos proporcionará un criterio de parada, pero en la mayoría de problemas es aún más importante la definición lo más precisa posible del límite superior, que además de la poda de dominios nos permite abandonar ramas del árbol de búsqueda y retroceder (*backtrack*) tan pronto como el coste asociado a las variables ya asignadas lo supere. Por lo tanto, el algoritmo de cálculo de límite superior puede representar una enorme diferencia en el tiempo transcurrido hasta que encontremos nuestra primera solución factible. Como es natural, una vez hayamos encontrado una solución factible, el límite superior pasará a ser el coste de ésta.

Otra estrategia muy común en CP es la organización del árbol de variables, siguiendo el método '*fail early*'. El concepto básico es que si nos estamos moviendo por una rama no factible o que no nos ofrece mejora, cuanto antes lo descubramos menos cómputo desperdiciaremos, por lo que se recomienda situar las variables 'difíciles' (de más coste o más centrales en el modelo de restricciones) al comienzo del árbol, de tal forma que si hemos asignado un valor satisfactorio a esas variables sea más probable que nos hallemos en una rama más satisfactoria de la búsqueda.

Cuando creemos nuestro modelo de CP debemos tener también en cuenta la búsqueda de simetrías, esto es, encontrar las variables que sean totalmente intercambiables para evitar todas las soluciones equivalentes que no nos aportan mejora. Una vez detectada una simetría debemos romperla para simplificar nuestro problema. El método más habitual para romper una simetría es establecer una nueva restricción, por ejemplo $X>Y$, de tal forma que se imposibilitan las soluciones equivalentes, sin embargo el algoritmo para detectar las simetrías y el como romperlas es especifico de cada problema por lo que hemos de ser cuidadosos en no eliminar posibles soluciones válidas en nuestro dominio.

Así pues, la creación del modelo del problema es muy específico para cada definición, siendo necesario en ciertos casos generar varias estructuras de datos con diferentes representaciones del problema o varias 'vistas' si se prefiere. Ciertas restricciones serán más efectivas o más fáciles de expresar si se aplican sobre una de las representaciones. Los cambios en una de las representaciones producirán cambios en la otra, en lo que se conoce como '*channelled representations*'.

La creación de un motor de resolución de CP que tenga un buen rendimiento en comparativa al estado del arte es una tarea muy compleja y que requiere un profundo conocimiento de las matemáticas asociadas, siendo por lo tanto impracticable en nuestro contexto de demostraciones frecuentes. Como ejemplo, el motor de CP usado en nuestro caso, *Choco*, fue desarrollado a lo largo de tres años por un equipo de cuatro personas, dos de ellas doctorados en CP. Por ello que se opta por la solución pragmática de escoger un *toolkit* ya desarrollado. Sin embargo, como se deduce de las estrategias de modelización comentadas, la construcción y optimización de un modelo adecuado a un problema concreto es una tarea ardua que requiere tanto el estudio detallado del comportamiento del problema, razonamiento sobre la propagación y tamaño de los dominios como ensayo y error sobre las posibles optimizaciones propuestas.

Todas las técnicas mencionadas anteriormente han sido usadas en nuestro caso.

3.2.2 Desarrollo

Como es natural, las principales variables a asignar de nuestro problema son las Reuniones (*Meetings*), mientras que la variable 'objetivo' sobre la que se lleva a cabo la optimización es *energyUsed*.

La primera de las vistas de nuestro problema es simplemente un vector con nuestras variables *Meeting*, que tienen inicialmente el dominio completo de acuerdo al calendario. Cada uno de nuestros *Meetings* tendrá inicialmente un dominio equivalente a todos las posiciones posibles (es decir la tabla bidimensional del calendario del problema * el número de habitaciones). Los argumentos son el 'nombre' de la variable y los extremos de su dominio.

```
meetingTable[i] = makeIntVar("slotMeeting" + meetingId, 0, totalSlots - 1);
```

Posteriormente, para los *Meetings* que tengan una duración superior a una ranura, crearemos *Meetings* virtuales, esto es, variables de *meeting* restringidas a:

```
model.addConstraint(eq(meetingTable[childPos],     plus(meetingTable[parentPos],
1)));
```

Contener el valor de su padre +1, esto nos ayudará a evitar el solapamiento de *meetings* en nuestro problema (simplemente aplicando la restricción *allDifferent* sobre nuestro vector de *meetings*) y también a calcular la energía operativa una vez tengamos el mapa tridimensional del que podemos deducir que *meetings* llevarán a cabo en habitaciones vecinas.

Nuestra segunda representación del problema es, como hemos comentado, un mapa tridimensional, con un *channeling* unidireccional dependiente de los valores del vector de *meetings*.

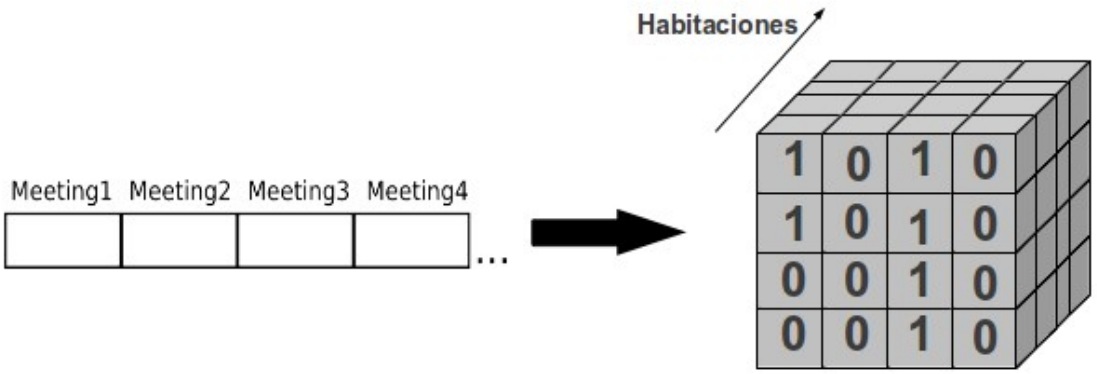

Figura 5 – Channeling entre ambas representaciones del problema

Siempre que uno de los *meetings* tome un valor en la solución parcial actual del árbol de búsqueda, la ranura asociada del cubo tridimensional será fijada a '1', en cualquier otro caso, la ranura tomará el valor de '0'.

Esta representación nos ayuda a asignar las restricciones de manera más eficiente: las restricciones de factibilidad se configurarán sobre el vector de la izquierda, reduciendo así el dominio de las variables *Meeting*, mientras que para el cálculo de la energía consumida por una solución concreta, nos es más natural pensar en términos espaciales en su representación a partir de un cubo, además de ser más eficiente como se ha experimentado a través de los anteriores modelos.

Hay que tener en cuenta que, pese a que cuando añadimos nuevas vistas para un modelo estamos añadiendo gran cantidad de variables, el motor de CP será instruido para únicamente generar el árbol e iterar sobre el vector de *Meetings*, por lo que estas variables adquieren valor de manera pasiva e inequívoca, convirtiéndolas en variables accesorias 'baratas' en tiempo de cómputo.

Para romper la simetría de nuestro problema tendremos que detectar no sólo variables *Meeting* que tengan el mismo dominio después de aplicar todas las restricciones, sino que contengan a su vez la misma lista de agentes asociados, de otra forma, aunque parezcan intercambiables a primera vista, la restricción de que un agente no puede estar simultáneamente en varios *meetings*, nos lleva a diferentes propagaciones. La simetría se rompe de la forma habitual comentada, válida en nuestro caso.

```
model.addConstraint(gt(meeting1, meeting2));
```

Nuestro programa nos avisará de las simetrías encontradas tras propagar las restricciones:

```
Symmetry slotMeeting8 [0, 359] slotMeeting12 [0, 359]
Symmetry slotMeeting12 [0, 359] slotMeeting13 [0, 359]
Symmetry slotMeeting6 [0, 359] slotMeeting7 [0, 359]
```

Como hemos comentado, por defecto, el motor CP escoge una distribución arbitraria del árbol de variables e itera los dominios de las mismas en orden creciente. Para nuestro problema comprobamos que era más eficiente iterar los dominios de forma '*greedy*', esto es, probar primero las posiciones del *meeting* que supongan menos coste energético. Por supuesto, colocar un *meeting* en su mejor posición no es definitivo, ya que como se puede deducir del problema, la solución óptima no tiene por qué tener ninguno de sus *meetings* en su particular posición óptima, pero es una aproximación que presenta grandes mejoras en la mayoría de los casos frente a una ordenación arbitraria. Para ello hubo que extender y modificar la clase *ValSelector* e implementar nuestra clase *MinimumCostValueSelector* mostrada en la figura 4.

De igual manera, hubo que generar una clase *BoundConstraints* extendida de motor de *Choco* que calculaba los límites superior e inferior de acuerdo al dominio de nuestro problema, teniendo en cuenta los *meetings* que quedaban por asignar, el coste de las habitaciones posibles y los vecinos que se podrían asignar en un caso óptimo.

Como se ha comentado anteriormente, el modelo aquí presentado es el tercero.

El primer modelo consistía en un único vector, sus elementos no eran *meetings* que tenían asignados posiciones, sino por el contrario las posiciones del calendario para todas las habitaciones que podían contener un número de *meetings* o estar vacías. A menos que el calendario y número de habitaciones fuese muy pequeño, el número de variables en el árbol de búsqueda era enorme. Se trataba de un modelo simple y fácil de entender pero que adolecía de graves problemas de rendimiento.

En el segundo modelo se adopta el vector de *meetings* tal y como lo hemos presentado, pero los cálculos energéticos se llevaban a cabo mediante complejas fórmulas que calculaban los vecinos sobre el vector plano. Como se comprobó más adelante, las restricciones complejas nos llevan a tiempos de propagación entre dominios enormes. Se decidió entonces el modelo con varias representaciones, donde las restricciones y cálculos de energía se pueden expresar con expresiones de *constraints* simples e intuitivas.

3.3 Algoritmo Iterated Local Search

3.3.1 Introducción

Como sucede frecuentemente, un algoritmo de búsqueda completa se vuelve intratable en memoria y en tiempo frente a problemas de gran tamaño, por lo que se decidió completar nuestra solución agregando algoritmos de búsqueda local que aproximasen una solución de buena calidad necesitando un tiempo de cómputo y memoria mucho más limitados. A costa, por supuesto, de no garantizar optimalidad. [14]

En nuestro caso, y tras estudiar las características del problema, se optó por una implementación del algoritmo de *Iterated Local Search* (ILS en adelante). Los métodos ILS son adecuados para escapar de óptimos locales y empezar el proceso de 'escalada de la colina' desde diferentes puntos del espacio de soluciones. Cuando se llega a un óptimo local, se intenta perturbar parte de la solución para encontrar un nuevo espacio inexplorado.

Este tipo de algoritmos son efectivos cuando tenemos una agrupación o *cluster* de óptimos locales a poca distancia entre sí, por lo que es posible introducir pequeñas modificaciones en un óptimo local que hayamos encontrado para desplazarnos hacia sus vecinos. Eventualmente, si en la agrupación de soluciones actual no conseguimos mejorar, se debe introducir una perturbación mucho mayor para encontrar una nueva agrupación de óptimos locales.

Gracias a los dominios obtenidos tras la propagación en el algoritmo de CP, se concluyó que, efectivamente, para muchos de los problemas propuestos la gran parte del espacio de soluciones viola alguna de las restricciones del problema, así como que las soluciones factibles a menudo están muy próximas entre sí. El propio algoritmo de CP genera típicamente 'ráfagas' de soluciones muy similares con un intervalo de cómputo grande entre ellas.

Los algoritmos ILS se diferencian pues de un *Repeated Local Search* en el concepto de 'memoria'. Un RLS comienza en un punto aleatorio en cada una de sus iteraciones y busca el óptimo local más cercano desde ese punto, un ILS recuerda su posición posterior y se mueve en referencia a esta.

3.3.2 Desarrollo

En la implementación se llevó a cabo un algoritmo de ILS -desarrollado “desde 0” en contraposición al algoritmo de CP- que solo contenía memoria de la última iteración. Se probaron versiones con varios tamaños de memoria -varias soluciones anteriores, se podía reiniciar desde cualquier de ellas-, pero no se obtuvieron mejoras apreciables conforme a la memoria inmediata de

14 http://sci2s.ugr.es/docencia/metaheuristicas/ILS.pdf (Ver Bibliografía en la última página)

la anterior solución y añadían complejidad al algoritmo, por lo que se decidió continuar con esta versión. También se intentó eliminar la memoria, creando efectivamente un RLS y, en este caso, el descenso de efectividad fue notable.

Nuestro algoritmo ILS se compone, pues, de diferentes partes intercambiables que nos permiten modificar su comportamiento aportando modularidad:

- Un criterio de aceptación: dado que solo tenemos memoria para nuestra última solución, el criterio de aceptación establece si es beneficioso saltar de nuestro estado anterior al nuevo estado encontrado. En nuestro caso el criterio de aceptación es simple, se cambiará el estado de partida si el nuevo estado tiene mejor coste. Recordemos que una solución con *n-1* violaciones de las restricciones siempre tiene mejor coste que una solución con *n* violaciones, por lo que este criterio nos hará progresar hacia soluciones factibles. La clase *BestCandidate* implementa este método y hereda de la clase abstracta *CandidateAcceptanceCriteria* usada para abstraer el criterio concreto en el núcleo del programa.
- Un generador de candidatos: los candidatos se generan asignando un valor a todas las variables no inicializadas. En nuestro caso tenemos una jerarquía de clases con una clase abstracta *CandidateGenerator* y dos implementaciones *RandomGenerator* y *GreedyGenerator*. Como su nombre indica, la versión '*greedy*' siempre asignará la posición de menor coste para la variable, mientras que la versión aleatoria asigna un valor al azar. Ambos se usan en nuestra implementación, por lo general el generador '*greedy*' es más óptimo a la hora de moverse dentro de un *cluster* de soluciones, mientras que el aleatorio nos impide volver a caer en el mismo *cluster* cuando queremos dar un salto grande en el espacio de soluciones.

Perturbación del óptimo local: como se ha comentado, para alcanzar un nuevo óptimo local debemos modificar el actual, para ello se destruye parte de la solución (asignaciones de *meetings*) y se reenvía la solución parcial al generador de candidatos para ser completada. Nuestra clase abstracta en este caso es *LocalOptimaScramble* y la implementación utilizada *IncreaseScrambleOnNoImprovement.* Tal como se deduce de su nombre, esta implementación lleva un conteo de los intentos de salto fallidos y aumenta proporcionalmente la parte de la solución destruida en función de estos, permitiendo escapar finalmente del *cluster* de soluciones actual.

Método de búsqueda local: una vez tengamos un candidato generado, intentaremos alcanzar el óptimo local más próximo, un método de '*hill climbing*' común a toda esta familia de algoritmos. En este caso contamos con la clase abstracta *LocalOptimaMethod* implementada por *IterativeVariableImprovement*. Esta clase tiene en cuenta todas las violaciones de las condiciones del problema y su algoritmo selecciona secuencialmente cada una de las variables. Partiendo de la posición actual de la variable *meeting* la reasigna a su posición de menor coste que no se superponga a ningún otro *meeting*. Contiene una lista de los *meeting* para los que es imposible mejorar, sin embargo hay que tener en cuenta que cada vez que se mueve uno de ellos es posible que sea factible mover otros. Cuando se alcanza un estado en el que es imposible mejorar el coste de ninguno de los *meetings*, se considera que se ha alcanzado el óptimo local, enviándose el resultado al criterio de aceptación.

Como veremos en la parte de evaluación, el estudio previo fue acertado y esta implementación permitió resolver problemas mucho más grandes que los tratables por el algoritmo

de CP. Al no necesitar un árbol de búsqueda y contener una memoria muy reducida, los nuevos candidatos progresivamente óptimos son generados a gran velocidad.

3.4 Algoritmo Genético

3.4.1 Introducción

Tras los satisfactorios resultados obtenidos con el método ILS, se decidió probar nuevos métodos de búsqueda local. Debido principalmente a su interesante concepto y la gran cantidad de código que era posible reutilizar del anterior método, se optó por implementar un algoritmo genético. [15]

Los algoritmos genéticos consisten en adaptar una versión simplificada del proceso de evolución natural al terreno de la optimización combinacional. Necesitamos por lo tanto implementar los mecanismo básicos de la evolución: la herencia genética, la mutación y la selección natural de los especímenes mejor adaptados a su contexto.

Una de nuestras primeras decisiones será, por lo tanto, decidir que representa el 'genoma' de cada uno de nuestros especímenes, el cual será objeto de la herencia y de mutaciones, que pueden ser aleatorias o no dependiendo que funcione mejor para nuestro problema específico. Otro factor fundamental es decidir cual será nuestra función de aptitud, es decir, cómo distinguiremos cuando uno de los especímenes es más apto que otros.

La herencia lleva asociada el concepto de selección natural y mutación, por lo que tenemos que decidir dos parámetros esenciales de nuestro algoritmo, la tasa de mutación y la tasa de selección. Escoger una tasa de mutación demasiado pequeña nos hará quedarnos atrapados con facilidad en un óptimo local, mientras que una demasiado grande impedirá una transmisión efectiva de la herencia, destruyendo los resultados conocidos. Un modelo similar se aplica a nuestra tasa de selección, por la que entendemos la posibilidad que tienen los 'mejores' padres de obtener descendencia. De la misma forma, si la tasa es demasiado alta, nuestras soluciones se copiarán sin las suficientes modificaciones, al no tener en cuenta otros candidatos que están cercanos a una buena solución. Si la tasa es demasiado baja, corremos el riego de no propagar adecuadamente nuestras mejores soluciones. Debemos obtener unos valores que nos permitan tanto mejorar progresivamente nuestras mejores soluciones como explorar nuevas partes aparentemente poco eficientes del dominio.

Una de las variantes más habituales en el contexto de los algoritmos genéticos es la recombinación, generando los nuevos especímenes mediante una combinación de dos o más especímenes anteriores (más mutación), imitando a los seres vivos sexuados. Usando este método se intenta propiciar que el nuevo espécimen adquiera las características más óptimas de los anteriores, siendo efectivo en más aspectos simultáneamente.

15 http://www.obitko.com/tutorials/genetic-algorithms/ (Ver Bibliografía en la última página)

3.4.2 Desarrollo

Vamos a desarrollar las decisiones tomadas para cada uno de los componentes básicos de un algoritmo genético mencionados:

- Genoma: en nuestro problema a tratar se toma la variables *meeting* y su dominio sobre la posición en el calendario y habitación como vector de genes.
- Función de aptitud: como es natural se usa la misma función de aptitud que en el caso del algoritmo ILS, los especímenes con menos violaciones de restricciones en primer lugar, y con menos coste en segundo, serán los más aptos.
- La tasa de mutación y tasa de herencia son parámetros que selecciona el usuario de acuerdo al problema concreto. Ambos son porcentajes, su funcionamiento se detallará en la descripción de las clases que lo implementan.
- Se implementan generadores de especímenes tanto de un único progenitor como de cruce entre dos progenitores.

Las clases más relevantes del algoritmo son las que heredan de la clase abstracta *GenerationConstruction*, *SingleParentGenerator* y *TwoParentsGenerator,* las cuales generan un nuevo vector de especímenes (*pool*) a partir de la anterior generación. En ambos casos la tasa de mutación se implementa de la misma forma: dado el porcentaje establecido por el usuario (real entre '0' y '1'), se genera un número aleatorio, si el resultado es menor que la tasa se destruirá el contenido de ese gen (variable *meeting*), si por el contrario el resultado es mayor se copiará intacta al nuevo espécimen. Tras este proceso reutilizaremos los generadores comentados en la parte de ILS (*GreedyGenerator, RandomGenerator*) para reconstruir la parte no asignada de este candidato.

Tras construir y evaluar la aptitud de una 'generación' de soluciones, estas son ordenadas de mayor a menor aptitud.

El método *SingleParentGenerator* implementa la herencia de la siguiente forma. Dada la tasa establecida por el usuario (real entre '0' y '1') se genera de nuevo un número aleatorio. Si el resultado está por debajo de la tasa, el 'padre' del nuevo espécimen será el que ocupe la posición 0 del vector (el más apto). Si el resultado está por encima, realizaremos una nueva tirada para el candidato en la posición 1 y así sucesivamente. En el caso de que ninguno de los candidatos sea seleccionado (muy poco probable si el *pool* es suficientemente grande y el factor está bien ajustado), el espécimen en la posición 0 será el progenitor. A modo de ejemplo, si escogemos un factor de 0.1, estadísticamente podemos esperar que el 10% de los 'hijos' sean del 'padre' 0, el 9% del padre 1, el 8.1% del padre 2, y así sucesivamente. De esta forma tenemos un método que premia a los candidatos más aptos, pero sin impedir la posibilidad de seguir explorando otras soluciones. El espécimen 0 siempre es copiado intacto por lo menos una vez a la nueva generación para evitar que perdamos definitivamente nuestra mejor solución.

La clase *TwoParentsGeneration* funciona de manera similar, pero seleccionando dos padres de la anterior generación, para lo cual se llevarán a cabo dos tandas del método anteriormente descrito, asegurando que los dos candidatos sean diferentes entre sí. La recombinación se lleva a cabo siguiendo un método aleatorio, donde la esperanza estadística se sitúa en una recombinación al 50% de los dos padres.

Como podemos extraer de la literatura asociada, uno de los problemas de esta familia de algoritmos consiste en que el criterio de parada no resulta evidente. En nuestro caso el usuario decidirá el número de generaciones a ejecutar, tras lo cual devolveremos nuestro candidato más apto.

Los resultados obtenidos con este algoritmo son menos satisfactorios que en el caso del ILS para algunas de las pruebas en la batería, en parte porque la recombinación no ofreció mejores resultados, lo cual es lógico teniendo en cuenta que casi todo el espacio de soluciones es no factible, por lo que combinando dos soluciones factibles probablemente empeoremos. Por otro lado, el algoritmo se queda atrapado en óptimos locales cuando existen grandes distancias entre los grupos de soluciones y no es capaz de saltar tan ágilmente como el ILS. Sin embargo, es notable la pequeña cantidad de código necesario para la implementación, debido también a la reutilización de clases del ILS, pareciendo un paradigma de programación elegante e ingenioso.

3.5 Interfaz web

3.5.1 Introducción

Como ya se ha mencionado, en el contexto de nuestro proyecto ITOBO teníamos frecuentes demostraciones para nuestros socios tecnológicos y, pese a que alguno de los métodos ya funcionaban satisfactoriamente, la resolución en modo textual era difícil de visualizar y apreciar por parte del público. Por ello se decidió crear una representación gráfica del problema y su solución. Dado que los algoritmos se ejecutaban a menudo en servidores de gama alta para mejorar los tiempos de respuesta, la implementación natural para tener acceso a estos servidores y evitar toda la problemática de la ejecución local eran las tecnologías web.

3.5.2 Desarrollo

La versión web se ajustó estrechamente al paradigma del prototipo evolutivo, comenzando por simples documentos HTML estáticos donde podíamos visualizar la solución, integrando progresivamente los métodos y herramientas necesarios para crear o modificar un problema y ,finalmente, superando en funcionalidad a la versión textual, ya que nos permite, a modo de ejemplo, visualizar las diferentes soluciones en orden de optimalidad de un algoritmo y descargarlas

en formato XML para su estudio. También podemos crear o modificar un problema existente de manera mucho más conveniente e intuitiva que la edición a mano de los documentos XML de entrada.

Junto con la evolución del prototipo, también fueron aumentando el número de tecnologías web utilizadas:

- *Javascript*: desde una fase temprana del prototipo se decidió que se adoptaría una interfaz similar a muchas aplicaciones nativas, donde el menú fuese fijo y el contenido de cada uno de estos menús variable, de tal forma que tenemos una parte de la página que se actualiza a petición del usuario gracias a funciones Ajax. Esto permite modularizar, ya que solo tenemos que reemplazar partes de la web mostrada y evita la reescritura de partes estáticas. Se utilizó extensivamente la librería de funciones accesorias *prototype*[16] para facilitar las llamadas a Ajax y modificar el DOM.
- *Spring* MVC: en estos primeros prototipos también se realizó una migración desde los *sciptlets* tradicionales a una versión modelo-vista-controlador. Este cambio supuso un acierto mejorando la modularidad y claridad del código y haciendo mucho más liviana la transferencia del código a otros desarrolladores. Como se ha mencionado anteriormente, el modelo ya se diseñó como una unidad independiente en la versión textual, por lo que hubo que crear únicamente los controladores, cuya lógica de programa resulta mayormente obvia en el contexto de nuestro problema (Añadir un agente, modificar un calendario, cambiar las propiedades térmicas de una habitación, etc) y la vista, lo cual supuso un reto estético más que técnico.

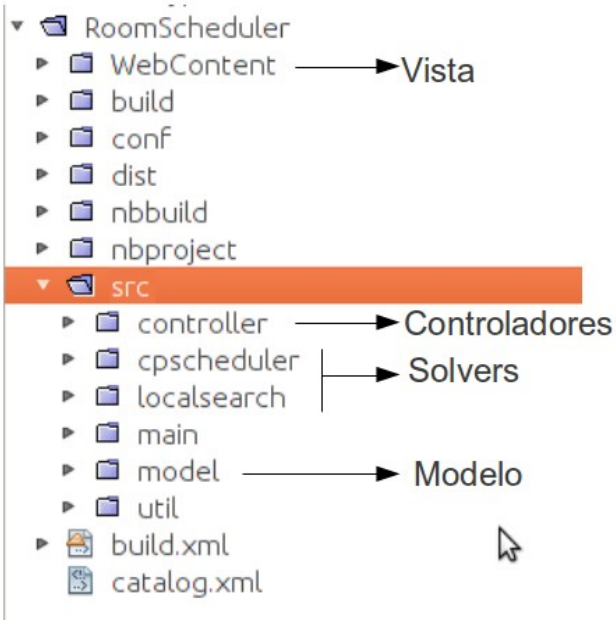

Figura 6 – Encapsulación del patrón MVC en el desarrollo web

16 http://www.prototypejs.org/

- CSS: para tratar de proporcionar una interfaz estéticamente agradable y coherente al usuario, se usaron hojas de estilo CSS. Se emplean algunas de las propiedades planificadas para CSS 3.0, como los bordes de sección redondeados.
- *Polling* asíncrono: mención especial merece el método utilizado para ir recogiendo y mostrando las sucesivas soluciones proporcionadas por los algoritmos, ya que en este caso necesitábamos actualizar la página de manera asíncrona y sin sobrescribir anteriores respuestas. A este efecto se estableció un mecanismo de *polling* periódico en Ajax, que recibía las actualizaciones en formato JSON, varios controladores conseguían transformar estas estructuras JSON en representaciones visuales de las soluciones que se organizaban por pestañas.

El aspecto y funcionalidad de la versión web serán desarrollados en profundidad en el anexo 'Manual de usuario de la web'.

3.6 Generador automático de problemas

A la hora de preparar la batería de pruebas de diferentes tamaños y características fue evidente que la generación manual era un proceso tedioso y propenso al error. En concreto, escribir problemas grandes que tengan por lo menos una solución factible puede ser extremadamente complicado si además deseamos que tengan una cantidad razonable de restricciones para evitar las soluciones triviales.

El generador hace uso extensivo de distribuciones gaussianas. La implementación original era linealmente aleatoria, sin embargo, se daban muchas instancias donde un *meeting* contenía a todos los agentes, *meetings* que solo contenían uno, habitaciones sin items, etc. El uso de distribuciones Gaussianas proporcionales al tamaño del problema produce ejemplares más útiles para la batería de pruebas.

Su funcionamiento básico consiste en la generación de *meetings* aleatoriamente y posterior asignación a una posición válida. Si no es posible colocar uno de los *meetings*, se generará de nuevo. De esta forma se garantiza que existe por lo menos una solución válida, sin generar soluciones triviales ya que el algoritmo de búsqueda no tienen ninguna información sobre esta. De hecho, hay casos en la batería donde el generador construyó un problema de tamaño excepcionalmente grande siguiendo este método, la comprobación de soluciones lo aceptó como válido, pero los algoritmos no fueros capaces de encontrar ninguna combinación factible.

4 Evaluación

Vamos a realizar la evaluación de nuestros algoritmos en función de varios criterios:

- Número de variables a resolver, o número de reuniones (*meetings*).
- Dominio de las variables, o lo que es lo mismo, tamaño del calendario por número de habitaciones.
- Restricciones en el problema derivadas de las restricciones en los calendarios.
- Comportamiento frente a un problema no factible.

Para aumentar la legibilidad y no extenderse excesivamente, se incluyen gráficas que resumen los resultados más relevantes, sin embargo se pueden estudiar los datos más detalladamente en la hoja de cálculo presente en la versión digital.

4.1 Comparativa 1 – Número de variables

En la siguiente gráfica podemos ver la mejor solución disponible y su coste (eje vertical) frente al número de variables del problema (eje horizontal). Podemos observar que el algoritmo genético sólo es capaz de llegar hasta el problema de 20 variables y el algoritmo ILS hasta el de 50 (sin embargo consigue una solución ligeramente mejor que la del algoritmo CP).

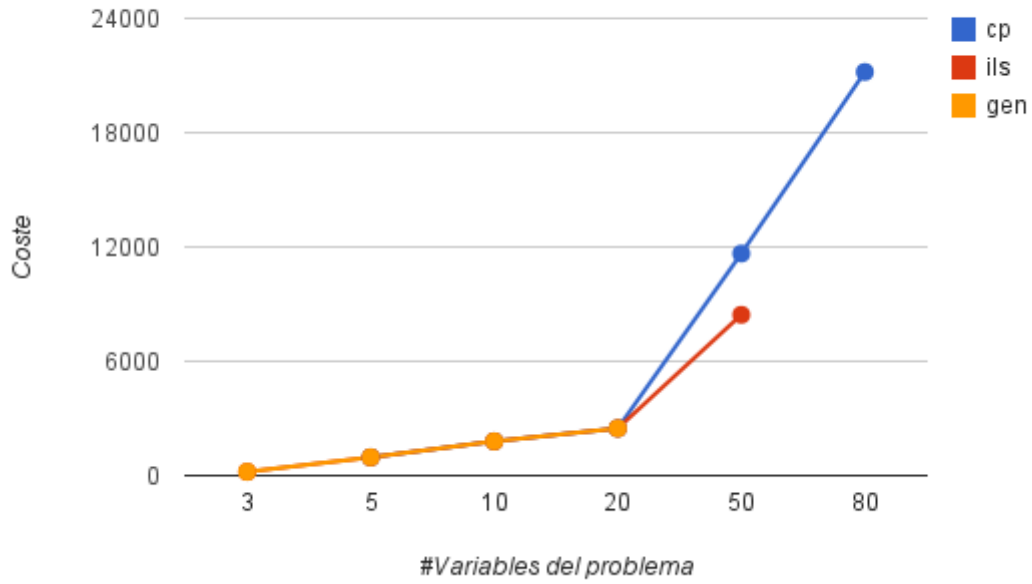

Figura 7 – Coste de la solución vs número de variables

En la siguiente gráfica podremos comparar los tiempos de cómputo que necesitaron cada uno de los algoritmos para obtener las soluciones de la anterior figura (tiempo en escala logarítmica)

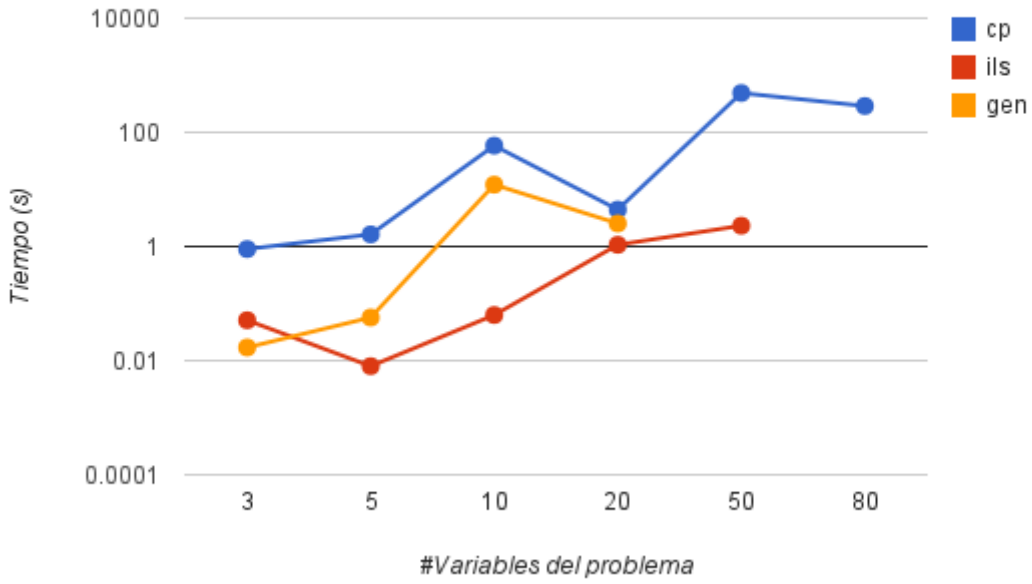

Figura 8 – Tiempo (log) vs número de variables

Podemos observar que el algoritmo de CP se comporta satisfactoriamente frente a grandes conjuntos de variables, el tiempo de arranque es notablemente mayor al del resto de algoritmos debido a la propagación previa, pero una vez terminado este proceso, cuenta con un árbol de búsqueda filtrado, lo que hace probable que encuentre soluciones válidas. Nuestro algoritmo ILS es capaz de igualar o mejorar la calidad de la solución en mucho menos tiempo, observamos, sin embargo, que ante el último problema fue incapaz de producir una solución debido al tamaño del espacio de soluciones y al alto número de restricciones. El algoritmo de CP garantiza optimalidad para los tamaños 3, 5, 10 y 20.

4.2 Comparativa 2 – Dominio de las variables

En la siguiente gráfica compararemos el coste de la mejor solución de cada uno de los algoritmos frente al dominio de cada una de las variables del problema.

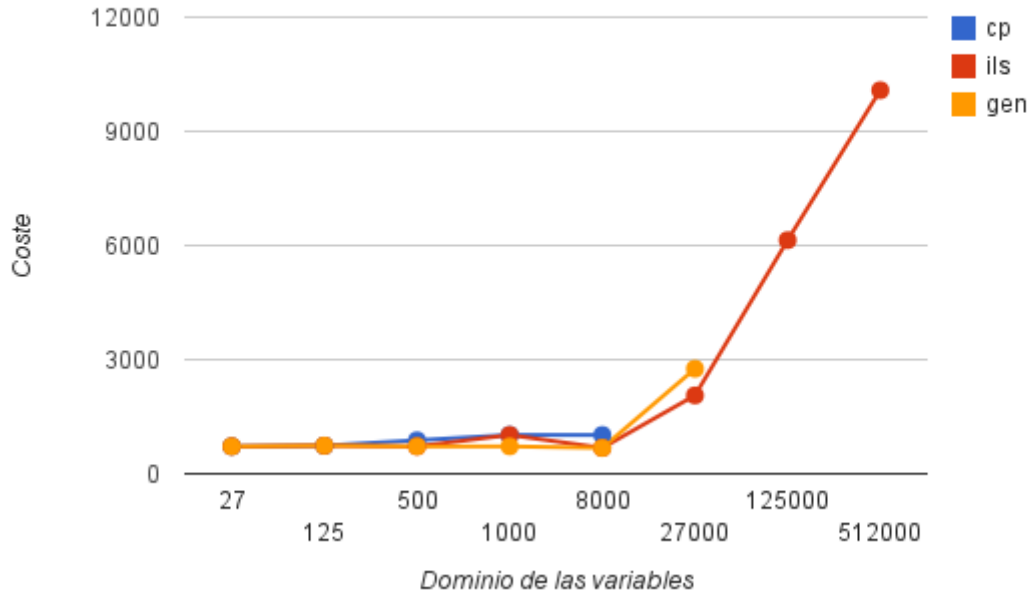

Figura 9 – Coste de la solución vs tamaño del dominio de las variables

Nuestro algoritmo de CP tiene problemas para manejar dominios muy grandes, lo que resulta lógico, ya que debe mantener el árbol de búsqueda en memoria y las propagaciones de restricciones durante la iteración se tornan exponencialmente caras. Los algoritmos de búsqueda local, al poseer una memoria de tamaño linear con respecto a la complejidad del problema – última solución en el caso del ILS y [*pool size* * *tamaño del problema*] en el caso del algoritmo genético –, resultan más adecuados para estos casos. En concreto, solo el ILS fue capaz de encontrar una

solución factible al problema con mayor tamaño inicial de los dominios: 512000 valores posibles para cada uno de los 30 *meetings*. CP garantiza optimalidad para los problemas con dominios de variable 27 y 125, se queda sin suficiente memoria para 27000.

4.3 Comparativa 3 – Cantidad de restricciones

Esta ocasión compararemos el comportamiento de los diferentes algoritmos frente a problemas con diferente número de restricciones sobre los dominios. En primer lugar la gráfica de comparativa de coste frente al número de restricciones.

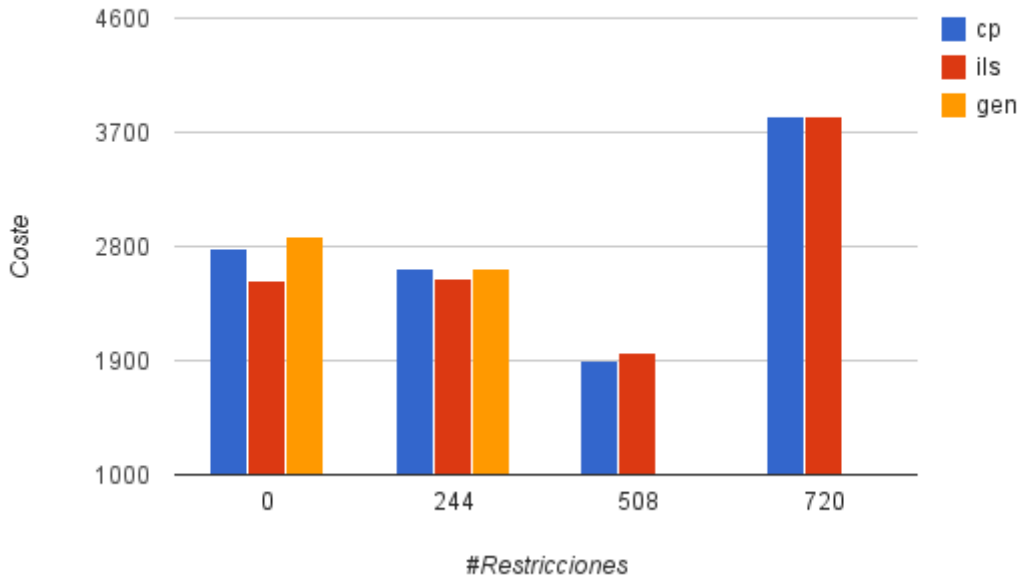

Figura 10 – Coste de la solución vs número de restricciones

En segundo lugar, el tiempo requerido para cada una de estas soluciones

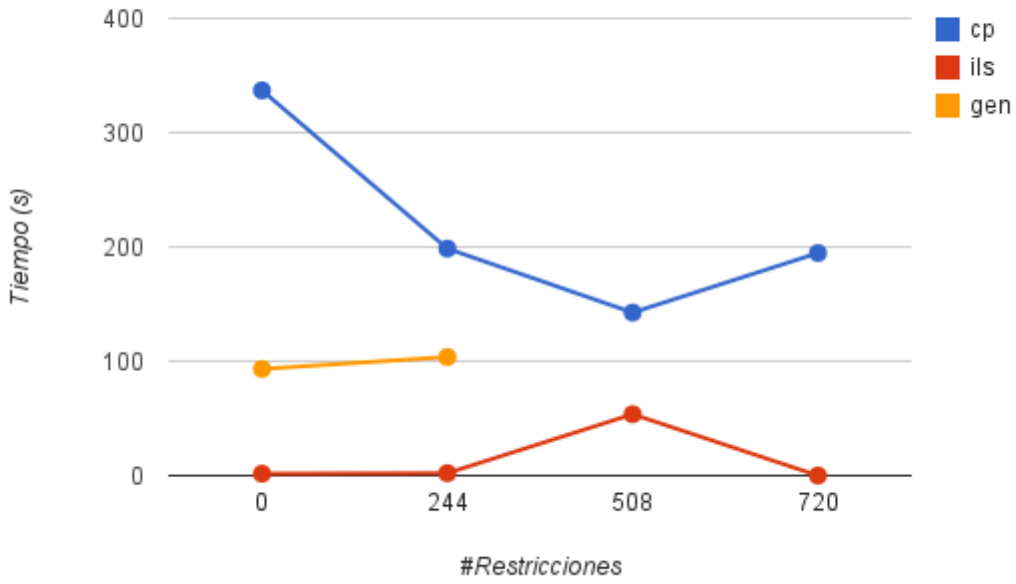

Figura 11 – Tiempo vs número de restricciones

Observamos el satisfactorio rendimiento de nuestro algoritmo de CP al aumentar el numero de restricciones que, pese a sufrir tiempos de 'propagación inicial' grandes, una vez terminado este proceso, posee un árbol de búsqueda de gran calidad, siendo capaz de asegurar el óptimo en los problemas muy restringidos (508 y 720). El rendimiento del ILS es muy aceptable y, en algunos casos, llega a igualar al algoritmo de CP. La búsqueda ILS es capaz de encontrar una buena solución, pero a menudo inferior a la óptima. El rendimiento de nuestro algoritmo genético se ve seriamente perjudicado en problemas muy restringidos, no pudiendo escapar de ciertos óptimos locales.

4.4 Comparativa 4 – Prueba de no factibilidad

En este caso usaremos solamente el algoritmo de CP, ya que es la única búsqueda completa y por lo tanto el único algoritmo que puede **garantizar** que un problema no tiene solución. Observemos el tiempo necesario para llegar a esta conclusión para diferentes tamaños del problema medidos en número de variables.

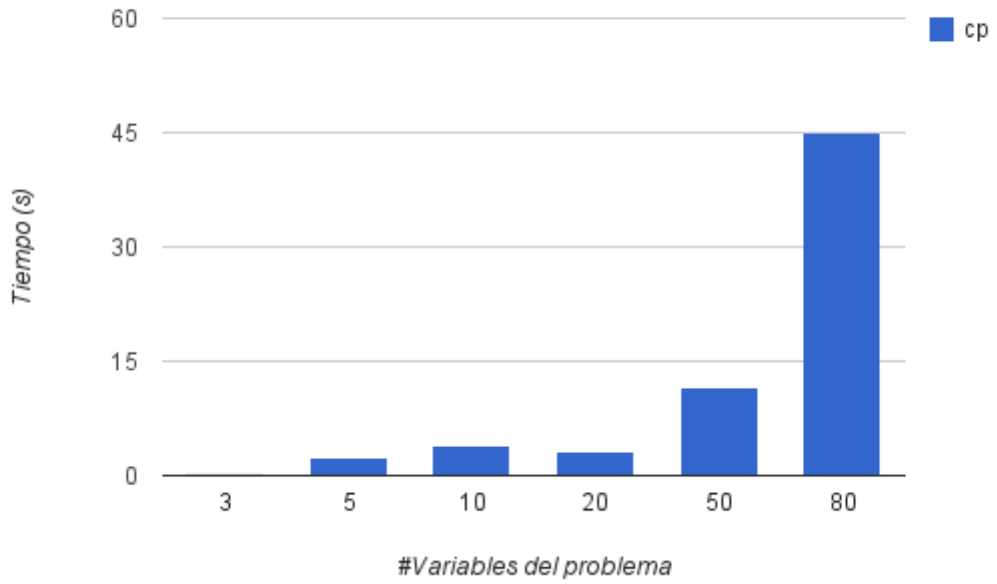

Figura 12 – Tiempo hasta detectar no factibilidad vs número de variables del problema

Observamos que el algoritmo de CP es capaz de encontrar una contradicción en sus restricciones, un problema no factible, con relativa facilidad incluso para problemas de gran tamaño. Resulta especialmente útil dado que es una información imposible de obtener con los otros dos métodos. Cabe la posibilidad de estudiar la factibilidad de un problema con el algoritmo de CP y, en caso que no se pueda demostrar su ausencia de soluciones, explorar las soluciones posteriormente usando ILS.

4.5 Conclusiones de las comparativas

Podemos inducir, pues, la topología de nuestro problema, como se ha comentado anteriormente, agrupada en *clusters* de soluciones separadas en el espacio. Las series de respuestas de los tres algoritmos a menudo contienen soluciones muy próximas en el coste y en el tiempo, mientras que se observan mayores cambios en la calidad tras grandes periodos de cómputo, sugiriendo que el algoritmo ha encontrado un nuevo *cluster*.

A partir de los resultados experimentales, a modo de comparativa, podemos sugerir lo siguiente:

- Algoritmo CP: al ser una **búsqueda completa** es el único método implementado capaz de asegurar que una solución es la **óptima**, resultando el más satisfactorio para problemas de tamaño pequeño – medio. Es un algoritmo pesado computacionalmente y la propagación de

restricciones consume grandes cantidades de **memoria**, resultando impracticable para problemas con grandes dominios. Debido también a la propagación, un número grande de restricciones es incluso beneficioso para su rendimiento, siendo también el único **capaz de detectar tempranamente un problema no factible**.

- Algoritmo ILS: proporciona una versión mucho más **ligera** que el algoritmo CP. En un gran número de las pruebas realizadas nos proporciona el **mejor compromiso entre tiempo de cómputo y calidad** de la solución. Debido a su implementación es capaz tanto de explorar núcleos de soluciones próximas gracias a la selección '*greedy*' y su algoritmo de implementación local, como de detectar el momento adecuado para el abandono y saltar a una región inexplorada del espacio de soluciones.
- Algoritmo Genético: pese a que su rendimiento es razonable en comparación a los demás algoritmos para la mayoría de las pruebas, está limitado por la ausencia en muchos de los casos de un "camino evolutivo de mejora continuada", dado que los núcleos de soluciones factibles son escasos y a menudo no tienen ningún parentesco entre ellos. Su comportamiento sugiere que sería más adecuado para otro tipo de problemas, donde fuese fácil encontrar una solución factible – o que todas las soluciones fuesen factibles – pero existiese un amplio margen para la mejora progresiva.

5 Conclusiones

5.1 Descripción resumida

Se ha estudiado un problema de optimización combinatoria aplicado al terreno de la domótica y ahorro energético e implementado tres diferentes soluciones capaces de afrontar la gran mayoría de las instancias reales con resultados satisfactorios, muy por encima de la capacidad organizativa manual que se pretendía reemplazar o apoyar. También se ha desarrollado una interfaz web para mejorar la visualización y comprensión del problema y sus soluciones, ofreciendo una plataforma de demostración fácilmente accesible.

El proyecto es, hoy por hoy, parte constituyente de la plataforma ITOBO. Su presente versión se ha transmitido a otros desarrolladores, sin que existiesen mayores problemas en la comunicación o adaptación del mismo debido a su modularidad y flexibilidad.

Cabe destacar el alto grado de satisfacción mostrado hacia el proyecto por parte de los socios que financian ITOBO, así como su despliegue en instalaciones reales.

5.2 Aportaciones

En opinión del autor, la principal aportación de este proyecto de fin de carrera es la adaptación de una serie de algoritmos presentes en la literatura de optimización combinacional al campo del ahorro energético, no existiendo en el momento de implementación del mismo ningún desarrollo que buscase los mismos objetivos. La optimización energética aplicada a los edificios inteligentes es un terreno con un amplio margen de mejora tanto en eficacia como en integración y modularidad, de acuerdo a la visión del consorcio ITOBO y del autor.

El presente proyecto, además de su aplicación práctica, presenta una serie de algoritmos que se complementan adecuadamente en la resolución de diferentes tamaños y características del problema, tal y como se ha discutido en la sección de comparativas, ofreciendo una plataforma de experimentación empírica que puede resultar didáctica y útil para personas interesadas en la materia.

5.2.1 Publicaciones académicas

Aunque, como ya se ha mencionado, el algoritmo de reserva de habitaciones se considera por el autor el cuerpo principal del proyecto, otro de los módulos desarrollados 'PVM Predictión tool', desarrollado en el Anexo B, arrojó unos resultados bastante satisfactorios, mejorando los encontrados en la literatura sobre el tema y dando lugar a dos publicaciones académicas que se

expondrán brevemente a continuación. Sus versiones completas serán incluidas en la copia digital de este PFC.

5.2.1.1 Predicting the desired thermal comfort conditions for shared offices

Artículo presentado en el congreso ICCCBE 2010 [17], organizado por la universidad de Nottingham. El congreso versa sobre la aplicación de nuevas tecnologías a la ingeniería civil y urbanismo. El artículo ha sido citado en cuatro ocasiones[18].

La publicación hace una mención inicial a los trabajos previos, basados en optimizar el valor de *PMV,* pero sin tener en cuenta ningún dato disponible sobre los usuarios en el pasado. Explica, mediante formulaciones matemáticas, el esquema de pesos descrito en la sección anterior y muestra las gráficas de resultados para las diferentes zonas climáticas, exponiendo las mejoras conseguidas en la predicción. Cabe destacar, también, que el método desarrollado necesita de menos sensores y menos lecturas diarias que las soluciones similares, representando a su vez un ahorro en materiales y despliegue.

5.2.1.2 Learning User Preferences to Maximise Occupant Comfort in Office Buildings

Presentado en el congreso IEA/AIE 2010[19]. Conferencia orientada a la aplicación de sistemas inteligentes en Ingeniería Industrial.

El desarrollo y conclusiones del artículo son similares a las del anteriormente mencionado. Sin embargo, en esta ocasión se experimenta con los diferentes pesos asignados a cada uno de los criterios mencionados anteriormente, estableciendo unos valores aconsejables y comportamiento del algoritmo en función de la cantidad de *feedback* que tenemos disponible y de la zona climática en concreto.

5.3 Cumplimiento de los objetivos

El autor entiende que los objetivos del proyecto se han cumplido con creces. Inicialmente se planteaba dar una solución satisfactoria mediante la familia de algoritmos CP prevaleciente en el laboratorio. Al alcanzar las limitaciones de este tipo de algoritmo, se ha llevado a cabo un nuevo proceso de investigación e implementación, dando lugar a un conjunto de soluciones que nos garantizan más herramientas capaces de cubrir casos no considerados inicialmente.

El desarrollo de una versión web tampoco se había considerado en la propuesta inicial del proyecto y, sin embargo, constituyó aproximadamente un tercio del esfuerzo, dando como resultado

17 http://www.engineering.nottingham.ac.uk/icccbe/
18 http://scholar.google.es/scholar?q=Predicting+the+desired+thermal+comfort+conditions+for+shared+offices
19 http://www.iea-aie2010.org/

una interfaz de apariencia agradable que hace uso de diversas tecnologías de la web dinámica y asíncrona.

5.4 Trabajo futuro

Se han propuesto diferentes mejoras al estado actual del proyecto por parte del autor, sus tutores y socios tecnológicos. Las más relevantes serían:

- Mejorar el algoritmo genético con varias *pools* que se sitúen en diferentes puntos del espacio de soluciones. Cuando uno de los *pools* no consiga mejorar en una cantidad de iteraciones adaptada al tamaño del problema, se plantearía destruirlo y reconstruirlo de nuevo en otro punto aleatorio. Se sugiere que esta mejora representaría un paliativo ante la ausencia de caminos evolutivos que conecten los núcleos de soluciones.
- Algunos motores de CP soportan la "relajación de restricciones", lo que hace posible asignar prioridades a las restricciones. En caso de que se descubra que un problema de entrada es no factible en su representación inicial, el motor comenzará a descartar restricciones de menor a mayor prioridad, siendo capaz en muchos casos de producir una solución de compromiso. El usuario es informado de qué restricciones fueron desechadas.
- También se propuso asignar preferencias discretas para cada una de las ranuras del calendario de los agentes, pasando del modelo binario -disponible/ no disponible- a un modelo más ajustado a las preferencias y comodidad del usuario.

5.5 Incidencias y experiencia adquirida

La principal incidencia encontrada durante el desarrollo de este proyecto de fin de carrera fue el ensayo y error de sucesivos modelos que sugerían mejoras en teoría, pero fallaban al compararse empíricamente. Esta complicación fue asumida desde un primer momento y era esperable dado el carácter experimental y de adaptación a un problema específico, por lo que la implementación de un algoritmo independiente de comprobación, la batería de problemas de entrada y un sistema de control de versiones que permitiese comparar los cambios y regresar a una versión anterior si fuese necesario, fue crucial para evitar las regresiones.

Otra complicación a mencionar fue el desconocimiento de muchas de las tecnologías web implicadas en la interfaz por parte del autor, lo que requirió expandir el periodo de estudio e investigación de las soluciones y lenguajes apropiados al caso.

Las experiencias adquiridas más relevantes en opinión del autor son:

- La importancia de un código modular y autoexplicativo: siempre que se afronta una implementación es necesario tener en mente la posibilidad de perder el contexto tras un periodo de tiempo y la necesidad de que otros desarrolladores hereden el código. Ambos

casos se dieron. Prácticamente nada de lo que parece obvio en el momento de la implementación lo será para otros desarrolladores o para el propio autor tras un periodo de tiempo, por lo que hemos de asegurar un código modular con nombres de clases, funciones y variables lo más autoexplicativos posible. Si es imposible deducir la función e importancia de una parte del código leyéndolo brevemente, se debe reescribir.

- Un sistema de versiones y una política de *backups* es esencial para no perder el conocimiento histórico de la evolución de un proyecto. Más aún dado el contexto de optimización, dependiente del "ensayo y error" en multitud de ocasiones.
- La filosofía de programación "*release early, release soon*" podría parecer que aumenta la carga de trabajo en las fases tempranas del desarrollo, pero se ha comprobado que evita descubrir los errores cuando "ya es demasiado tarde", o se requiere una reescritura masiva para adaptar funcionalidades no previstas. Incluso a nivel de motivación personal, es mucho más positivo contar con una demostración parcial sobre la que otros programadores y los destinatarios del proyecto puedan ofrecer su opinión.

6 Anexos

Anexo A - Manual de usuario de la web

Menú principal

A continuación desarrollaremos el uso de la interfaz web, con ayuda de capturas de pantalla y los comentarios necesarios para empezar a emplear la herramienta, necesitando tan solo un navegador.

La pantalla principal tiene el siguiente aspecto:

Figura 13 – Pantalla principal

Podemos observar que disponemos de botones para crear un nuevo problema partiendo "desde cero", o bien cargar un fichero de problema existente. Los botones de guardado y ejecución no estarán disponibles hasta que no contemos como mínimo con los elementos "Agentes, Habitaciones y Reuniones".

Supongamos que tenemos un problema contenido en un fichero que deseamos ejecutar:

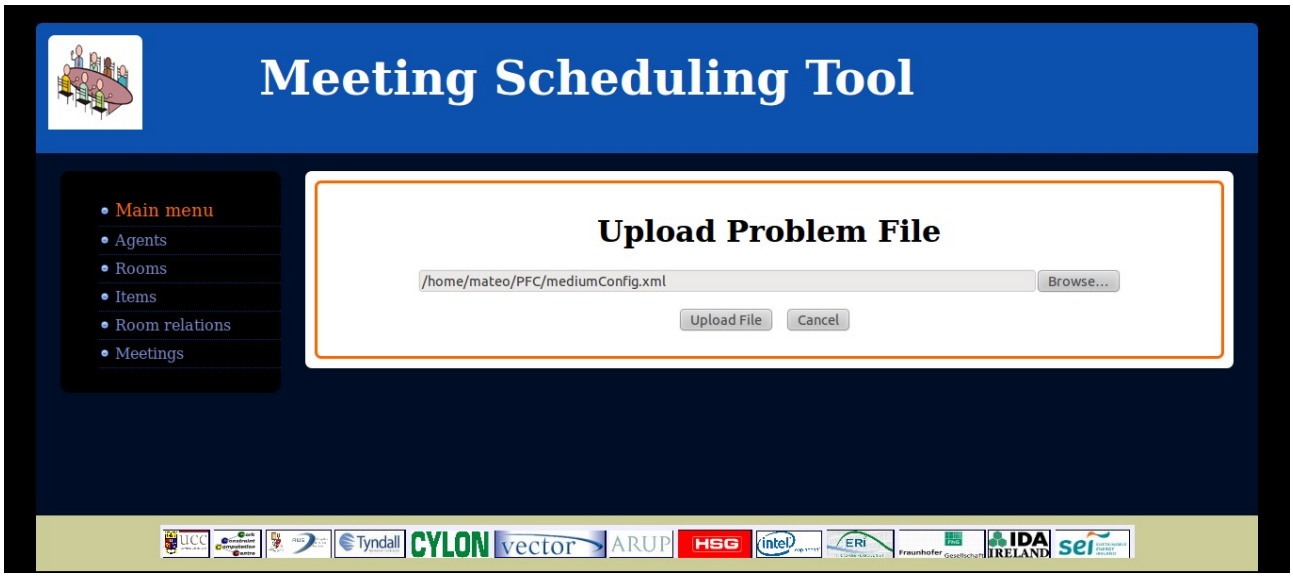

Figura 14 – Cargando un problema en memoria

Una vez cargado el problema, su nombre se nos mostrará en la parte superior izquierda de la interfaz. Podemos ver las diferentes secciones del menú que serán ahora accesibles:

Figura 15 – Menú de entidades del problema

Las diferentes entidades a las que podemos acceder son los Agentes, Habitaciones, Items, Relaciones entre habitaciones (referentes al modelo de transferencia de calor) y Reuniones. Como veremos más adelante, podemos visualizar los elementos presentes, editarlos o añadir nuevos.

Agentes

Para nuestro problema de ejemplo vamos a acceder al menú de Agentes:

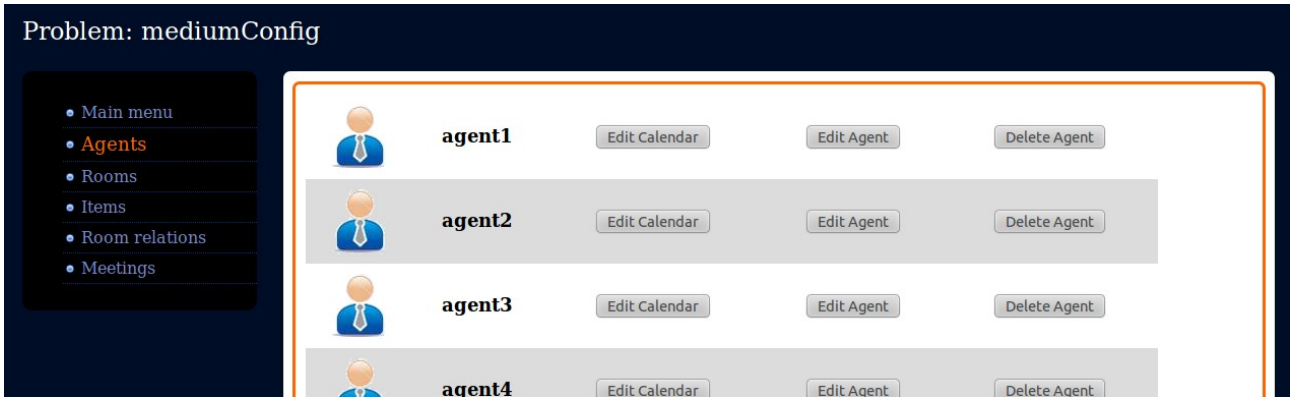

Figura 16 – Menú de agentes

Podemos observar los diferentes agentes que forman parte de nuestro problema. Como se ha explicado en la definición del mismo, un agente es una de las entidades disponibles para nuestras Reuniones. Podemos editar su calendario personal, editar el nombre del agente o eliminarlo.

Al final de la página observamos que es también posible definir un nuevo agente, simplemente asignándole un nombre y editando posteriormente su calendario privado.

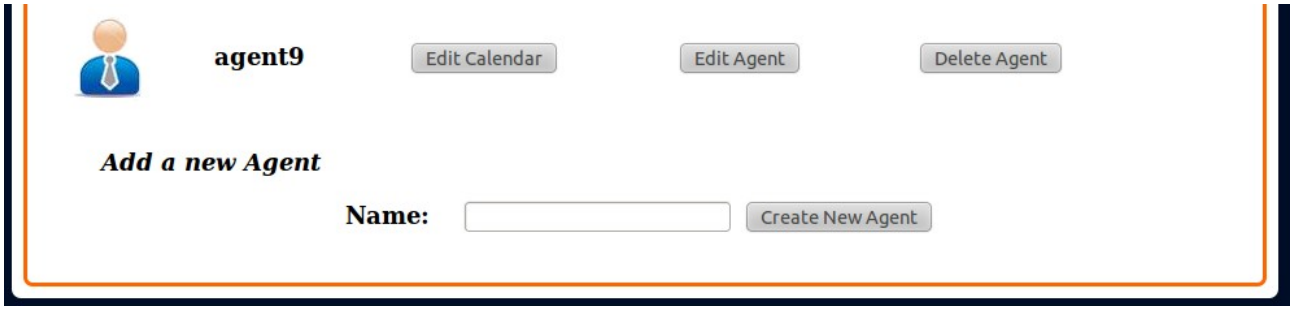

Figura 17 – Añadir un nuevo agente

Estudiemos, pues, la interfaz desde la que podremos configurar el calendario privado de un agente mediante el boton "*Edit Calendar*":

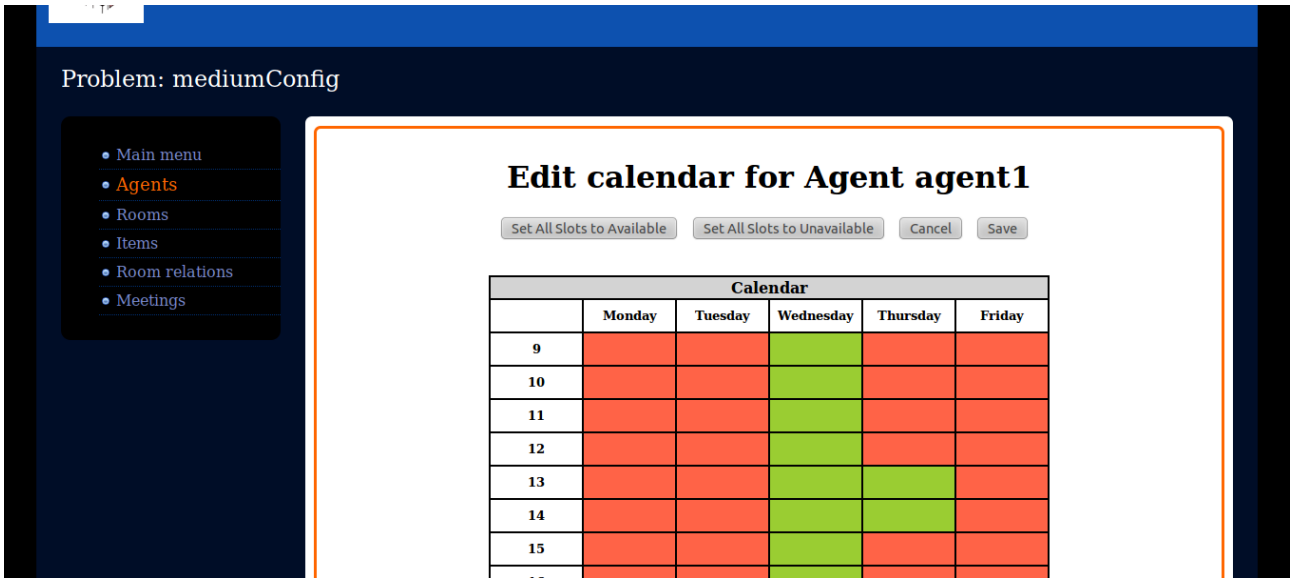

Figura 18 – Editar calendario de un agente

Como podemos deducir de la imagen, cada una de las casillas del calendario representa una ranura de tiempo. Las filas y columnas están etiquetadas de acuerdo a los nombres asignados en el momento de creación del problema. Para el caso particular de este problema, usaremos los días laborables de la semana y un horario de trabajo de 9 de la mañana a 8 de la noche. Las casillas verdes indican las posiciones de tiempo en las cuales el agente está disponible para una reunión. Podemos *clickarlos* individualmente para cambiar su estado, usar los botones en la parte superior para designar todo el calendario como disponible o no-disponible y, adicionalmente, podemos *clickar* en la etiqueta de una fila o columna para cambiar el estado de todas las ranuras asociadas.

Una vez hayamos terminado de editar el calendario, es posible guardar o cancelar nuestra configuración. El formato generado siempre intenta comprimir la información, de tal forma que tendrá en cuenta si hay mayoría de ranuras de un tipo, o si una fila contiene ranuras del mismo tipo.

Si eliminamos un agente es posible que alguna de nuestras reuniones se haya quedado vacía. El sistema intenta mantener la coherencia del problema en todo momento, por lo que lo eliminará de la misma, informando al usuario

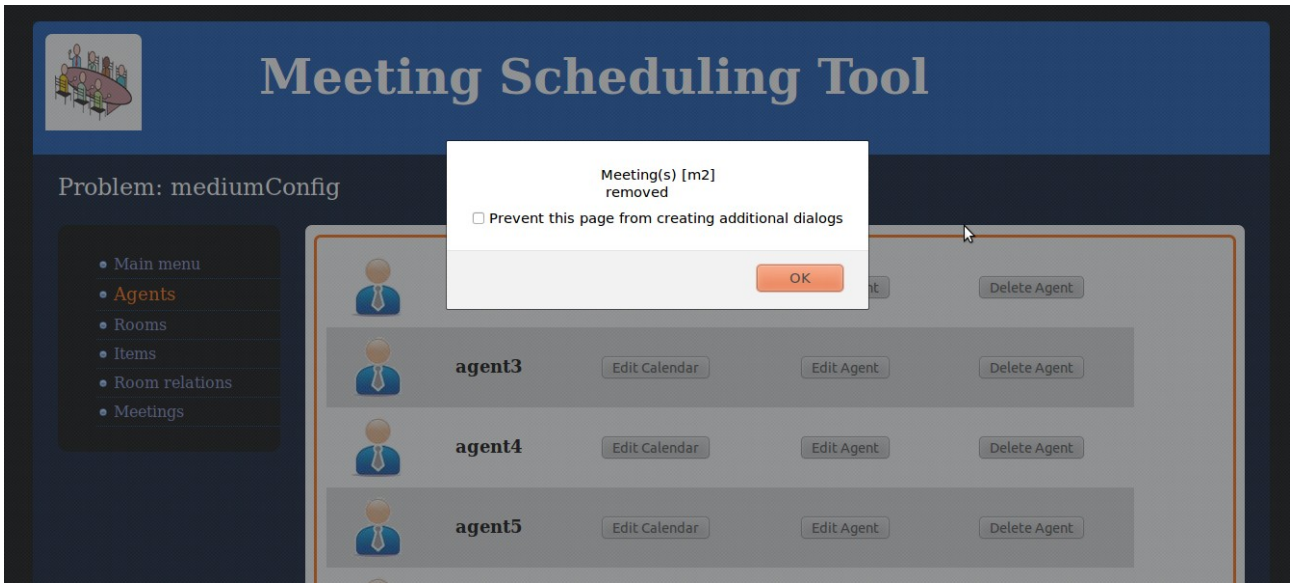

Figura 19 – Aviso de eliminación de reuniones vacías

Habitaciones

Accediendo al menú de habitaciones podemos encontrar lo siguiente:

Figura 20 – Menú de habitaciones

Para cada una de las habitaciones tenemos la siguiente información, de izquierda a derecha:

- Nombre de la habitación.
- Máxima ocupación aceptable.

- Energía para calentar la habitación desde un estado de reposo o "*Startup Energy*". Esta energía no se tendrá que utilizar si una reunión se realiza en la misma habitación para la ranura de tiempo inmediatamente anterior.
- Energía para mantener la habitación caliente durante una ranura de tiempo o "*Operating Energy*". El coste indicado disminuirá si tienen lugar reuniones en el mismo espacio de tiempo en habitaciones contiguas.
- Lista de material contenido en esta habitación que puede ser requerido para las reuniones, como veremos más adelante.

Podemos en todo momento editar una de las habitaciones y cambiar cualquiera de los parámetros mencionados:

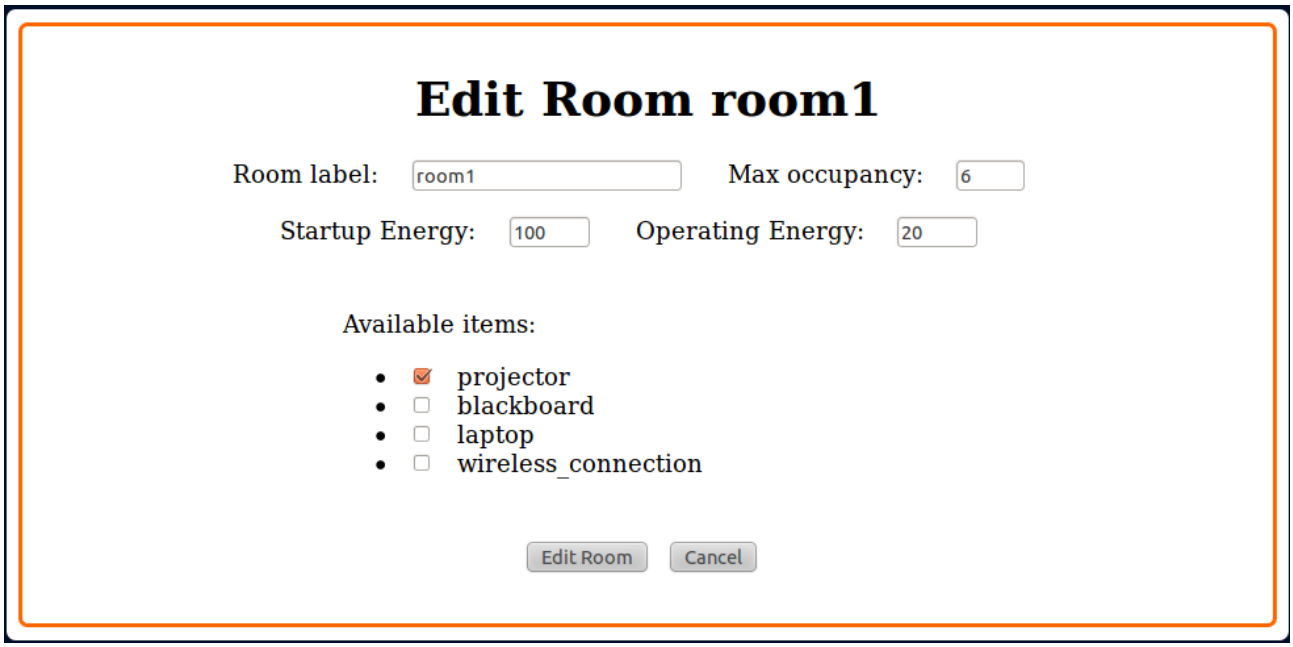

Figura 21 – Editar habitación

De manera análoga a los agentes, las habitaciones tienen su propio calendario privado que puede ser editado:

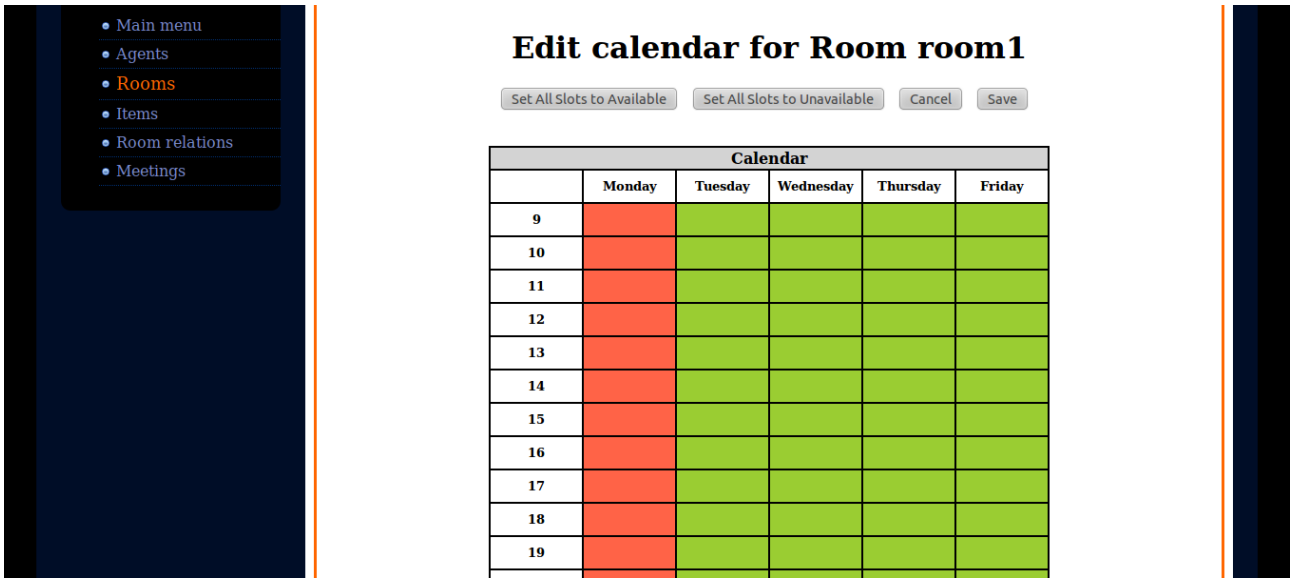

Figura 22 – Editar calendario de habitación

Y podemos añadir nuevas habitaciones con:

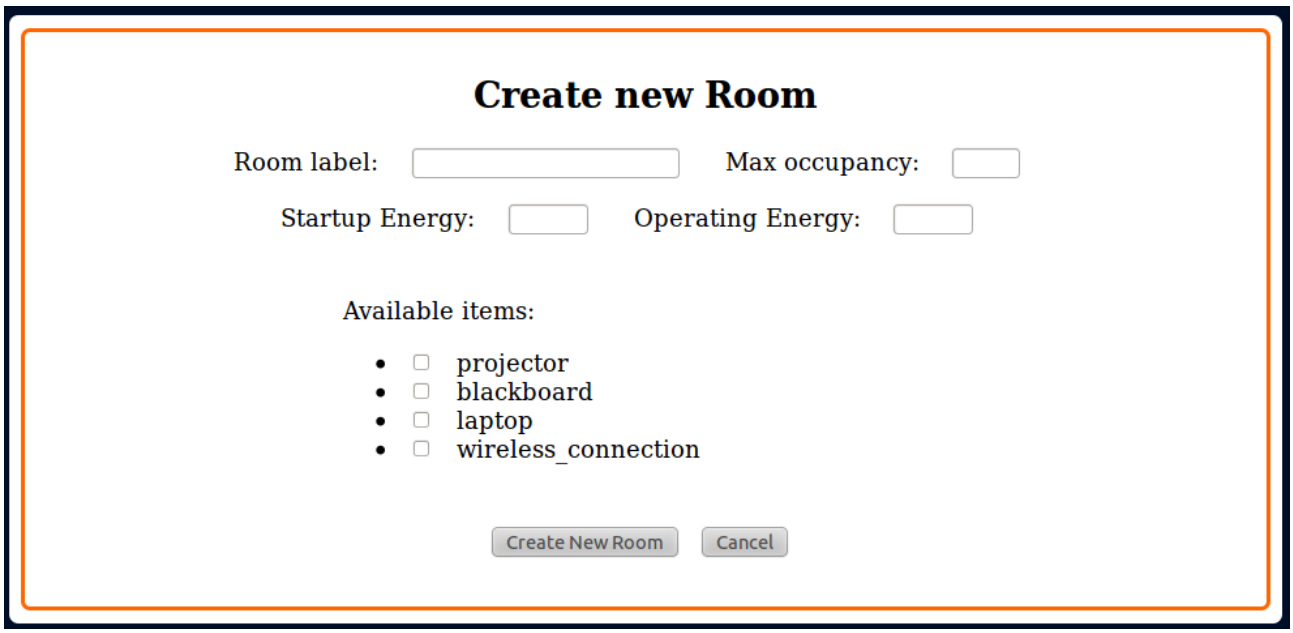

Figura 23 – Crear nueva habitación

Items

El menú de los *items* es el más simple de todos, debido en parte a que no son entidades por sí mismos sino, más bien, características atribuibles a reuniones y habitaciones. En cualquier caso, desde su sección podemos crear nuevos *items* o eliminar los existentes:

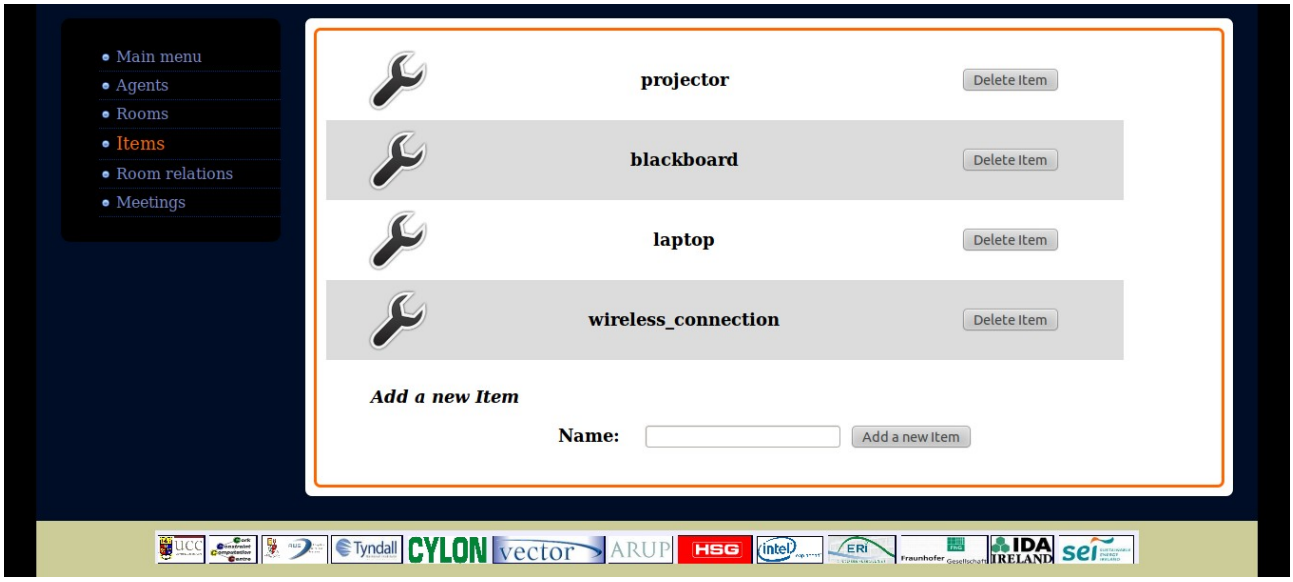

Figura 24 – Menú de items

De nuevo, el sistema intentará mantener la coherencia en todo momento, así que si eliminamos un *item*, será eliminado de los requisitos de las reuniones y del material disponible en las habitaciones.

Room Relations

Las relaciones entre habitaciones, o "*Room Relations",* expresan la transferencia de calor de acuerdo al modelo energético comentado anteriormente. En su sección del menú encontraremos:

Figura 25 – Menú *Room Relations*

Por cada una de las relaciones tenemos los siguientes elementos:

- Nombre de la primera habitación contigua.
- Nombre de la segunda habitación contigua.
- Porcentaje de energía operativa, "*Operating Energy",* que la segunda habitación ahorrará si la primera se encuentra en funcionamiento simultáneamente. A modo de ejemplo, si la segunda habitación consume 20 unidades energéticas, al aplicar una transferencia del 10% pasará a consumir 16.
- Porcentaje de energía operativa, "*Operating Energy",* que la primera habitación ahorrará si la segunda se encuentra en funcionamiento simultáneamente.

Asimismo, podemos editar cualquiera de las relaciones o crear nuevas:

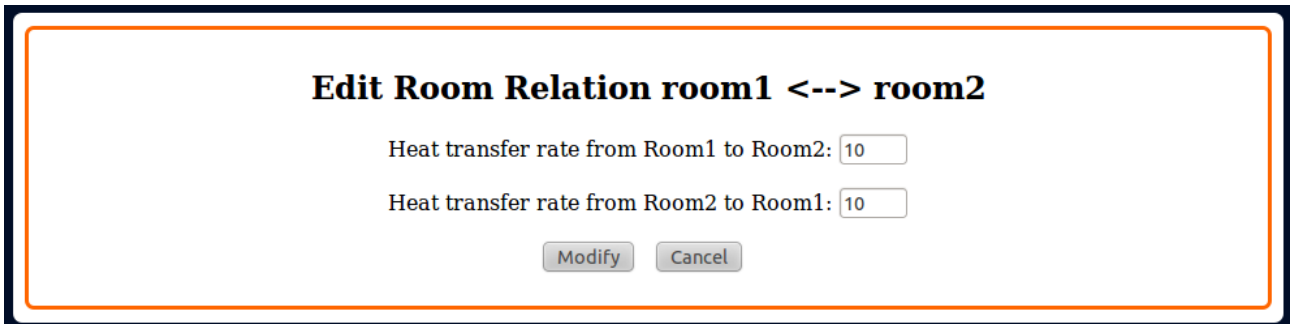

Figura 26 – Editar relaciones energéticas entre habitaciones

Para mantener la coherencia, el sistema nos avisará si la relación entre habitaciones es no factible, esto es, si es posible que la energía consumida por una habitación sea inferior a 0 gracias a la suma de las diferentes habitaciones contiguas, lo cual está prohibido en nuestro problema.

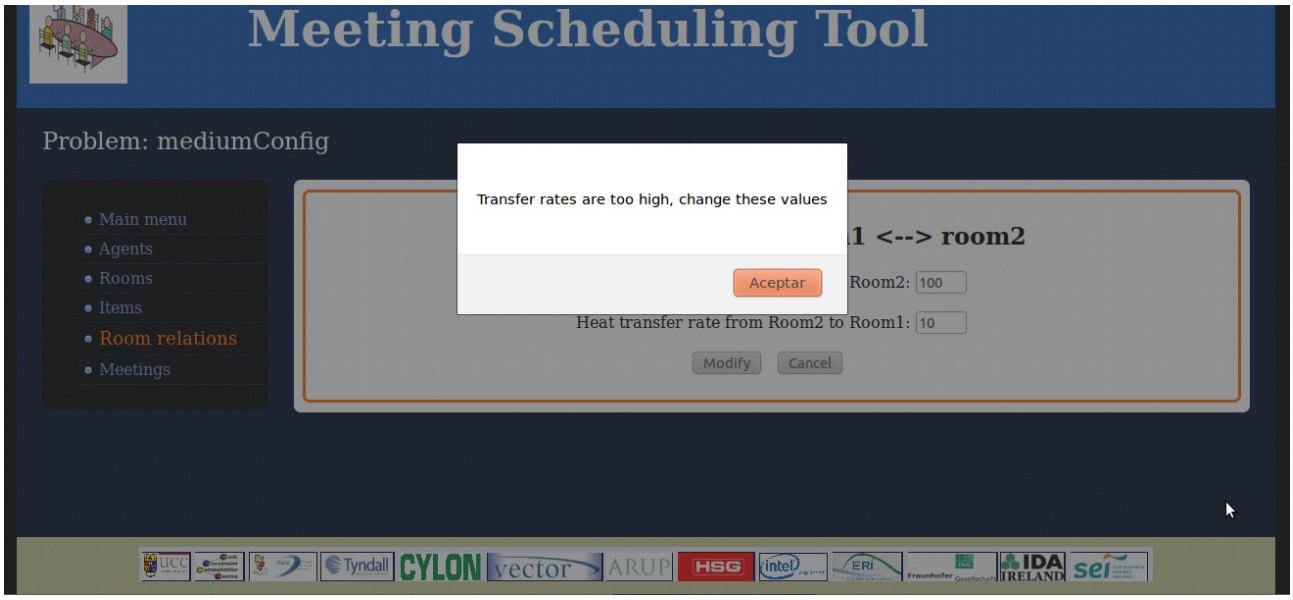

Figura 27 – Modelo energético no válido

El sistema también controla que no exista más de una relación configurada entre dos habitaciones dadas.

Reuniones

Las reuniones son la entidad central de nuestro problema y el objeto de la solución al mismo. En su sección podemos encontrar:

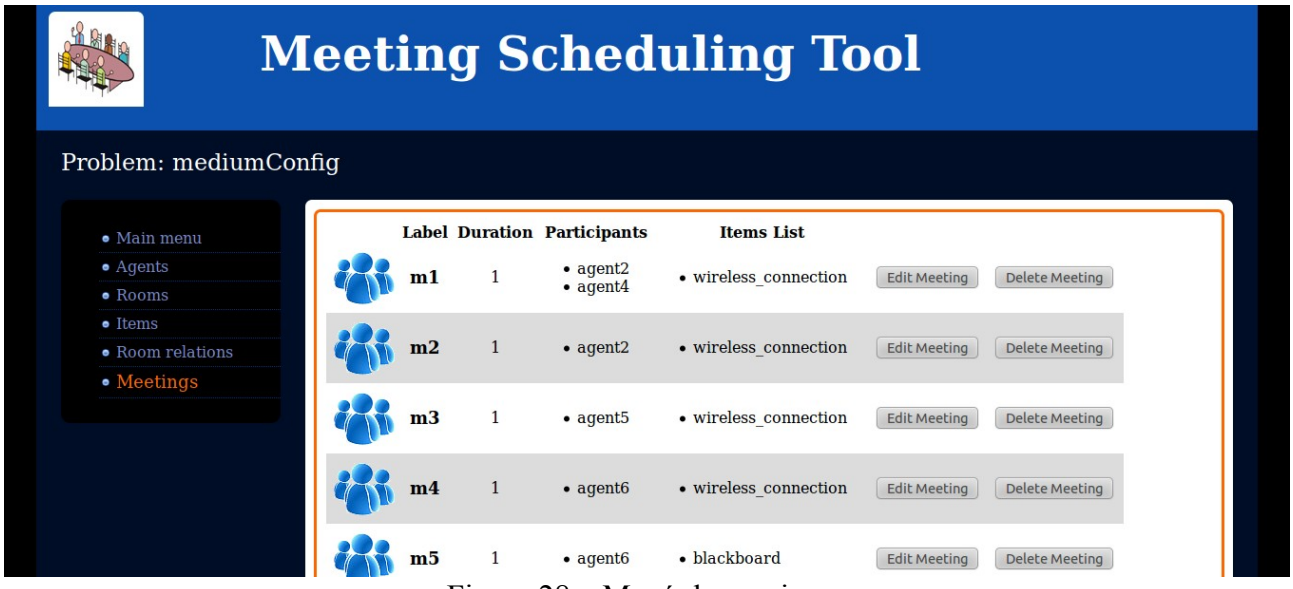

Figura 28 – Menú de reuniones

Para cada una de las reuniones tenemos la siguiente información:

- Nombre de la reunión.
- Duración medida en número de ranuras.

- Lista de agentes participantes.
- Material requerido por esta reunión.

Es posible editar cualquiera de nuestras reuniones, así como crear otros nuevos:

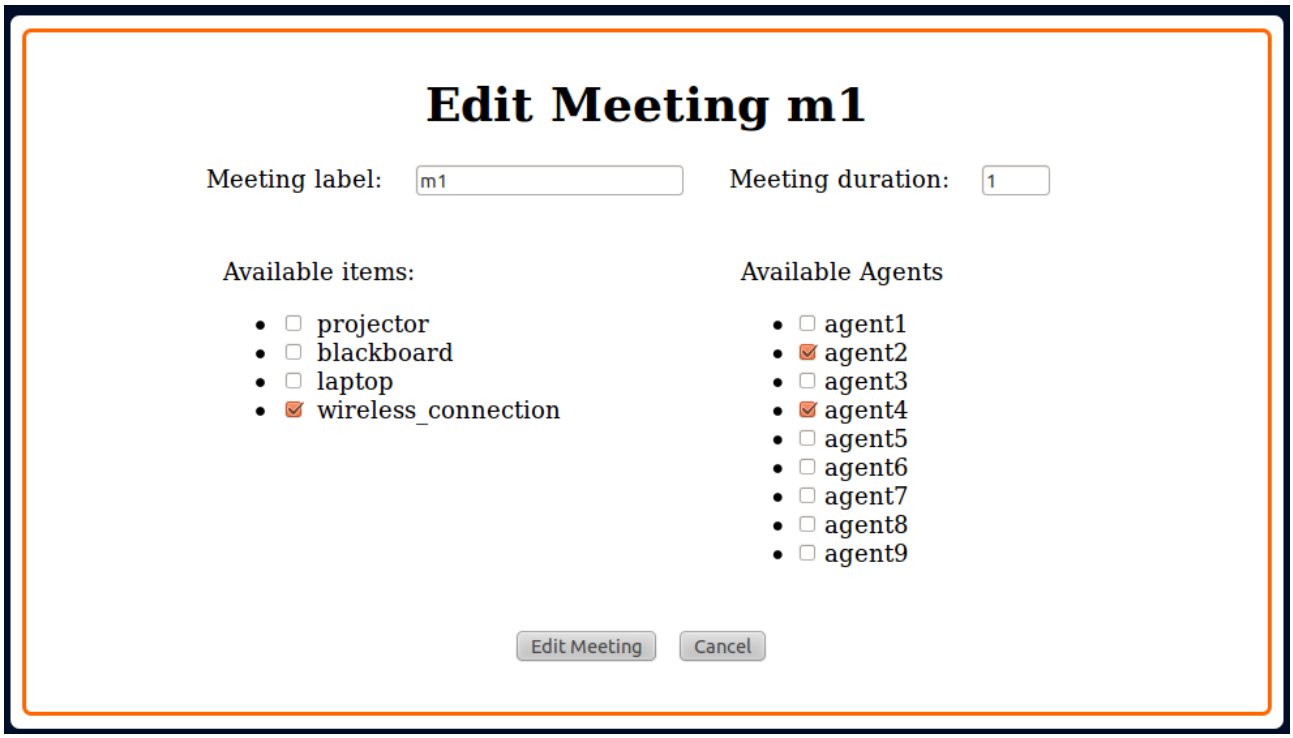

Figura 29 – Editar una reunión

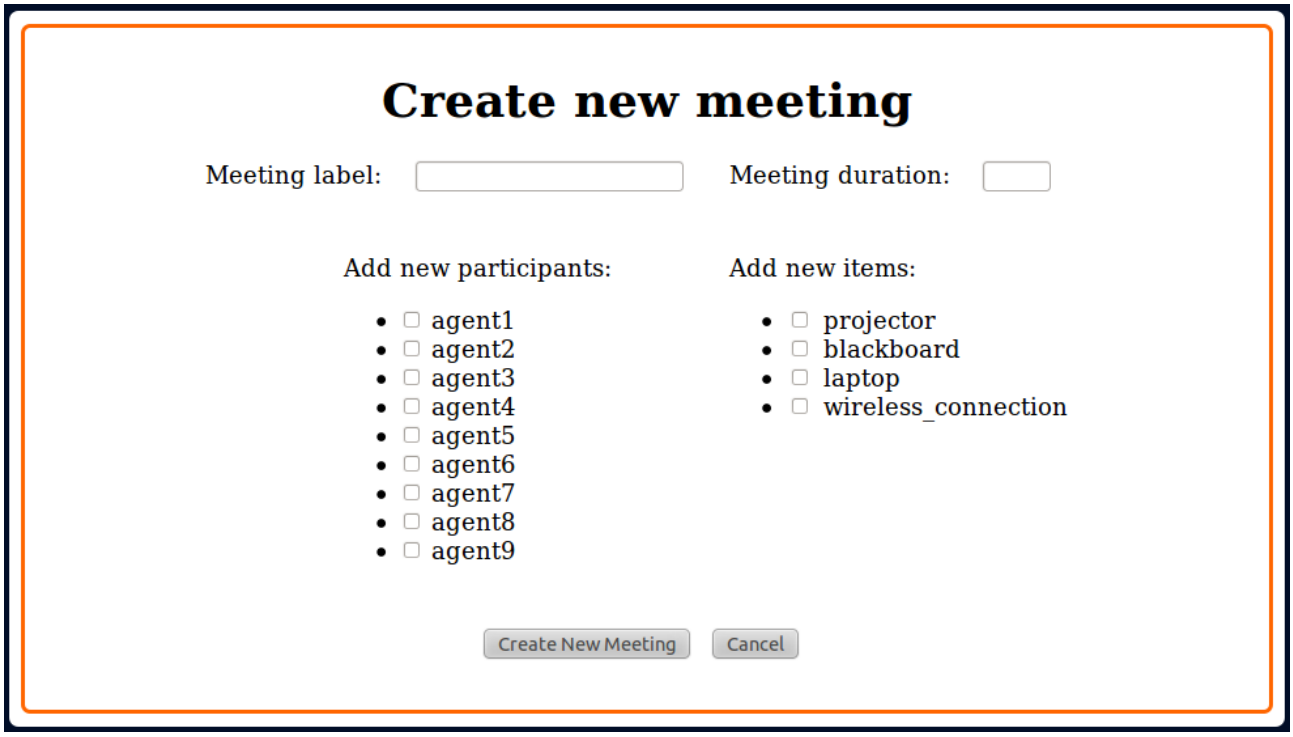

Figura 30 – Crear nueva reunión

Menú de ejecución

Una vez hayamos editado, creado o importado el problema que deseamos resolver, accederemos al menú de ejecución mediante el botón "*Execute Problem*", ya observado en el menú principal:

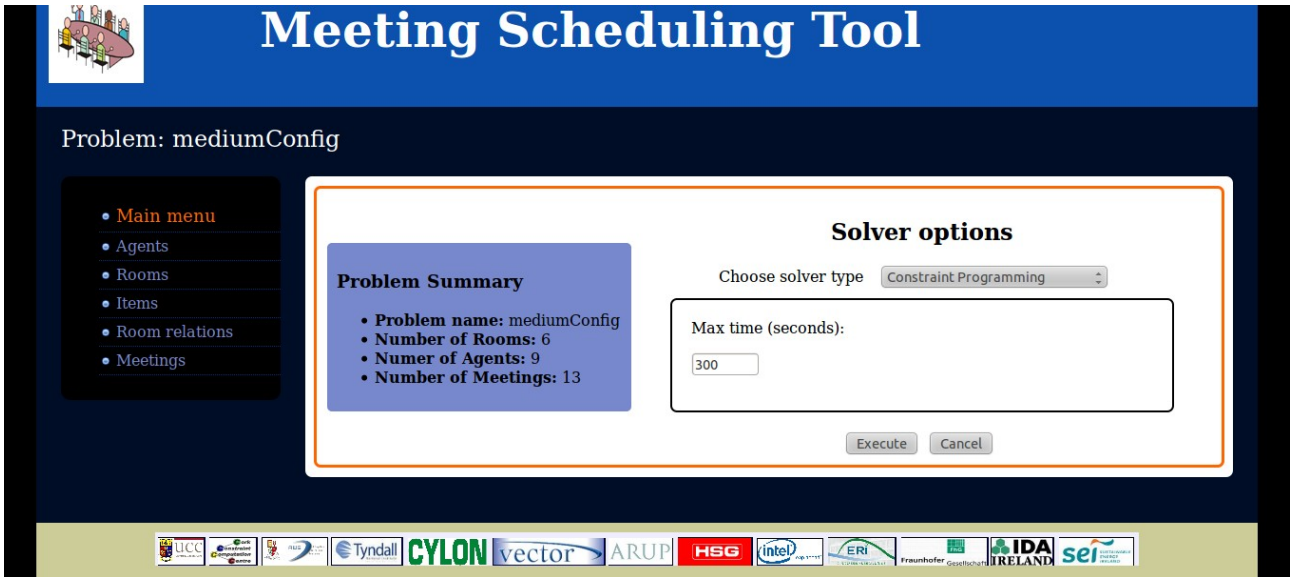

Figura 31 – Ejecutar problema

En primer lugar, podemos ver un breve resumen del problema a ejecutar en el cuadro situado a la derecha del menú principal: nombre del problema, numero de reuniones, agentes y habitaciones.

En la parte derecha podremos elegir que tipo de *solver* deseamos utilizar, cada uno de ellos asociado con un conjunto de parámetros.

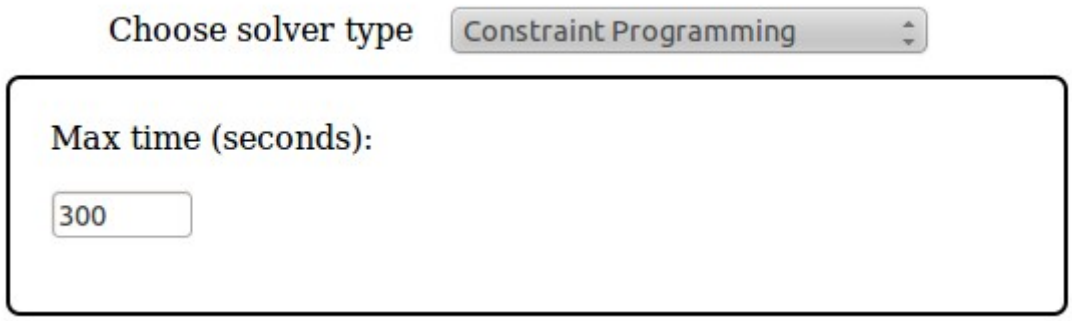

Figura 32 – Algoritmo CP

El algoritmo basado en *Constraint Programming* nos ofrece la opción de establecer un tiempo máximo de búsqueda. Resulta de utilidad debido a que el algoritmo encuentra frecuentemente soluciones factibles y satisfactorias, pero la prueba de optimalidad puede ser demasiado costosa para nuestro caso.

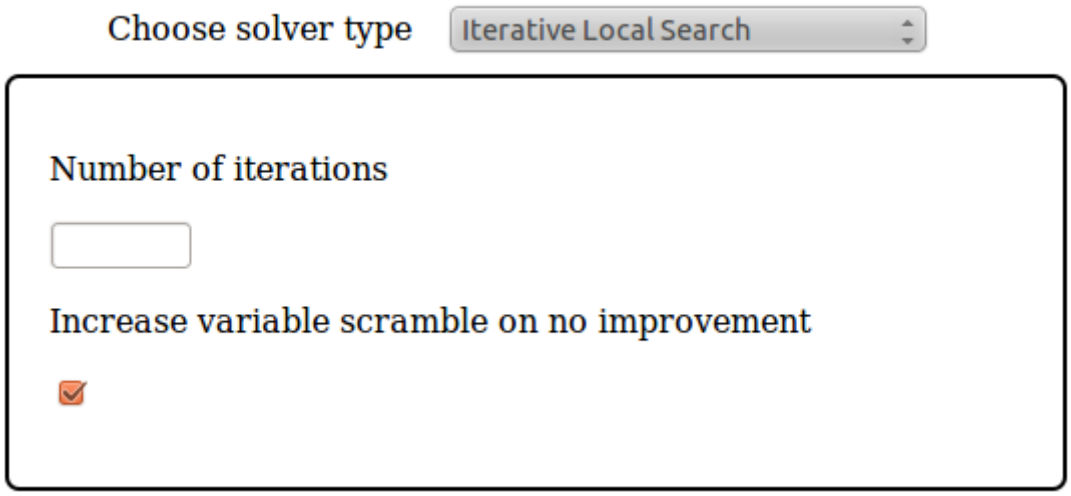

Figura 33 – Algoritmo ILS

El algoritmo ILS nos ofrece dos parámetros:

- El número de iteraciones de búsqueda local a realizar sobre el problema.
- Opción para aumentar el *scramble* o perturbación de la solución actual en caso de que no mejoremos nuestros resultados, a menudo necesaria para escapar de óptimos locales.

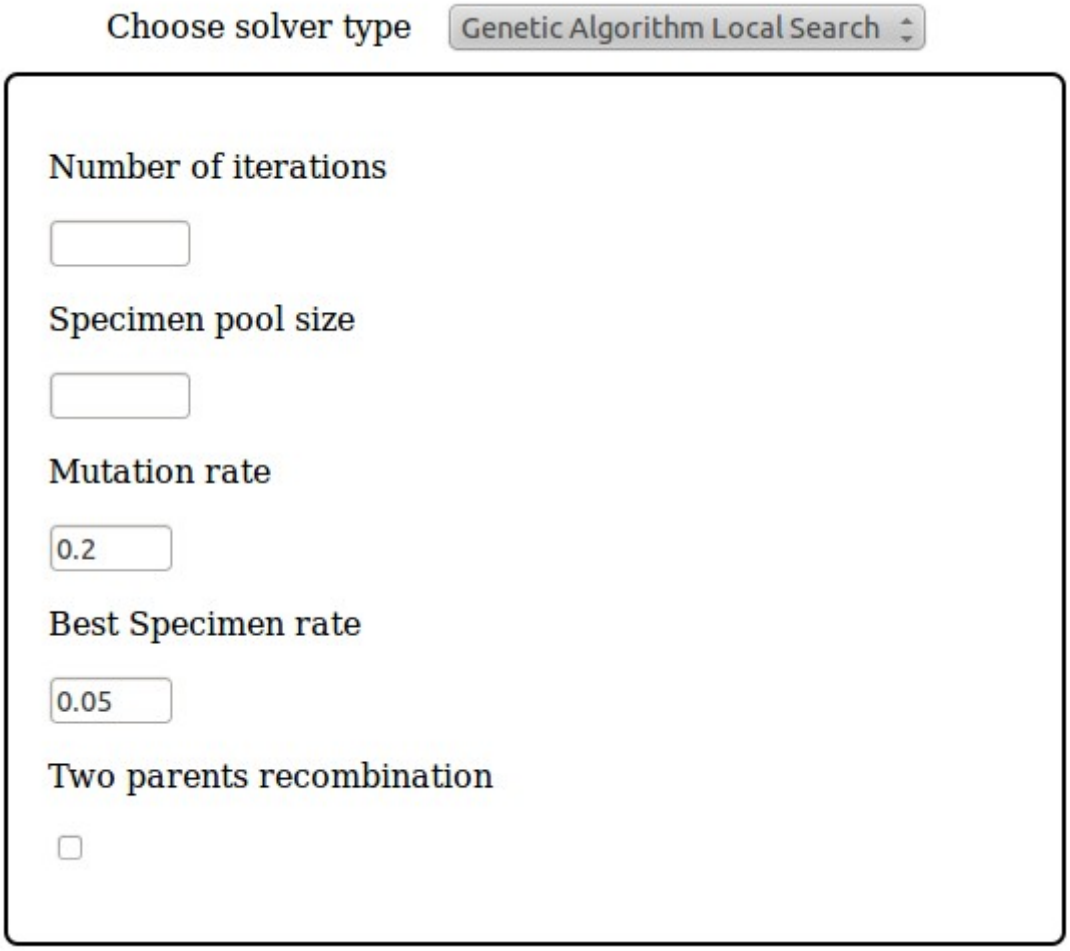

Figura 34 – Algoritmo genético

Las opciones disponibles para este algoritmo son:

- Número de iteraciones, o lo que es lo mismo, número de generaciones de especímenes a desarrollar hasta la terminación.
- Tamaño en número de especímenes de cada una de las generaciones.
- Ratio de mutación: media estadística del porcentaje de genes que serán alterados cuando reciba la herencia de su predecesor.
- Probabilidad de herencia del mejor predecesor, cuyo funcionamiento ha sido explicado en la sección pertinente al algoritmo genético.
- Combinación de dos predecesores. Si esta casilla está marcada, se combinarán dos predecesores de la anterior generación para crear el espécimen, en caso contrario solo se utilizará uno.

Una vez hayamos configurado el algoritmo a utilizar y sus parámetros, comenzaremos la ejecución del problema pulsando en el botón "*Execute"*:

Interfaz de representación de soluciones

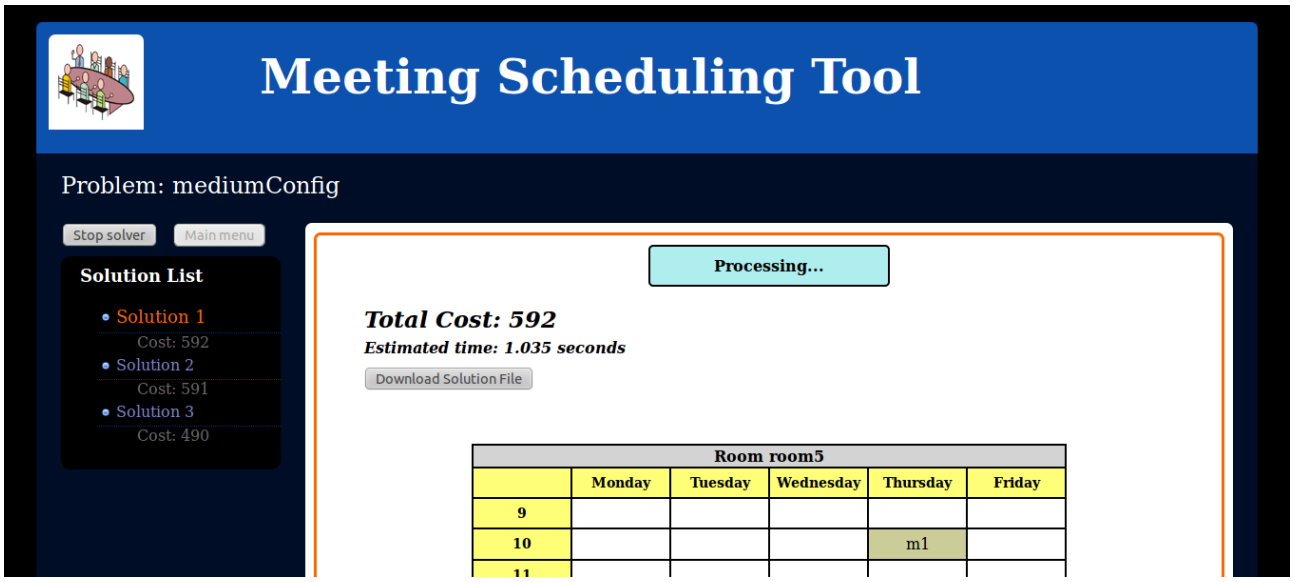

Figura 35 – Menú de ejecución

Tras lanzar la ejecución, nos encontramos con el menú de la Figura 35. A la izquierda tenemos la lista de soluciones encontradas actualmente, con su coste asociado. Podemos *clickar* en cualquiera de ellas para mostrar la solución gráfica completa. Encima de la lista tenemos dos botones, *"Stop solver"* para detener el procesamiento del algoritmo y "*Main menu*" para volver al menú principal. Solo podemos volver al menú principal si el procesamiento ha terminado o ha sido detenido por el usuario.

En la parte superior del cuadro blanco vemos un área informativa que podrá tomar los siguientes valores:

- *Processing:* el algoritmo se encuentra aún en ejecución y buscando soluciones.
- *Optimal solution found:* la búsqueda completa -CP- ha finalizado el recorrido por el árbol de soluciones y garantiza que la mejor solución presentada es la óptima.
- *Time out limit reached:* la búsqueda completa -CP- ha alcanzado su tiempo límite y se ha detenido.
- *Unfeasible:* la búsqueda completa -CP- ha descubierto una contradicción en las condiciones, por lo cual no es posible encontrar ninguna solución al problema.
- *Local optima method finished:* el método de búsqueda local ha llegado a su máximo de iteraciones, por lo que se ha detenido.
- *Solver stopped:* el algoritmo de búsqueda se ha detenido a petición del usuario.

Las soluciones alcanzadas presentan la siguiente interfaz:

Total Cost: 490

Estimated time: 0.313 seconds

Download Solution File

Room room6					
	Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
9					
10					
11					
12					
13					
14		m8			
15		m5			
16		m6			
17		m13			
18		m12			
19		m7			
20					

Room room5					
	Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
9					
10				m1	
11					
12					
13					
14		m2			
15		m3			
16		m4			
17		m11			
18		m11			
19		m9			
20		m10			

Figura 36 – Ejemplo de solución

En la parte superior de la Figura 36 podemos encontrar la información referente al coste de la solución, así como el tiempo necesario para calcularla. El resto de la interfaz nos muestra una representación gráfica de las habitaciones y ranuras utilizadas por cada uno de las reuniones, de acuerdo al calendario establecido para este problema. La solución escogida para el ejemplo es particularmente pequeña, con el fin de poder representarla entera en este documento.

Finalmente, utilizando el botón *"Download Solution File"*, podemos descargar una descripción en formato XML de la solución que estamos visualizando:

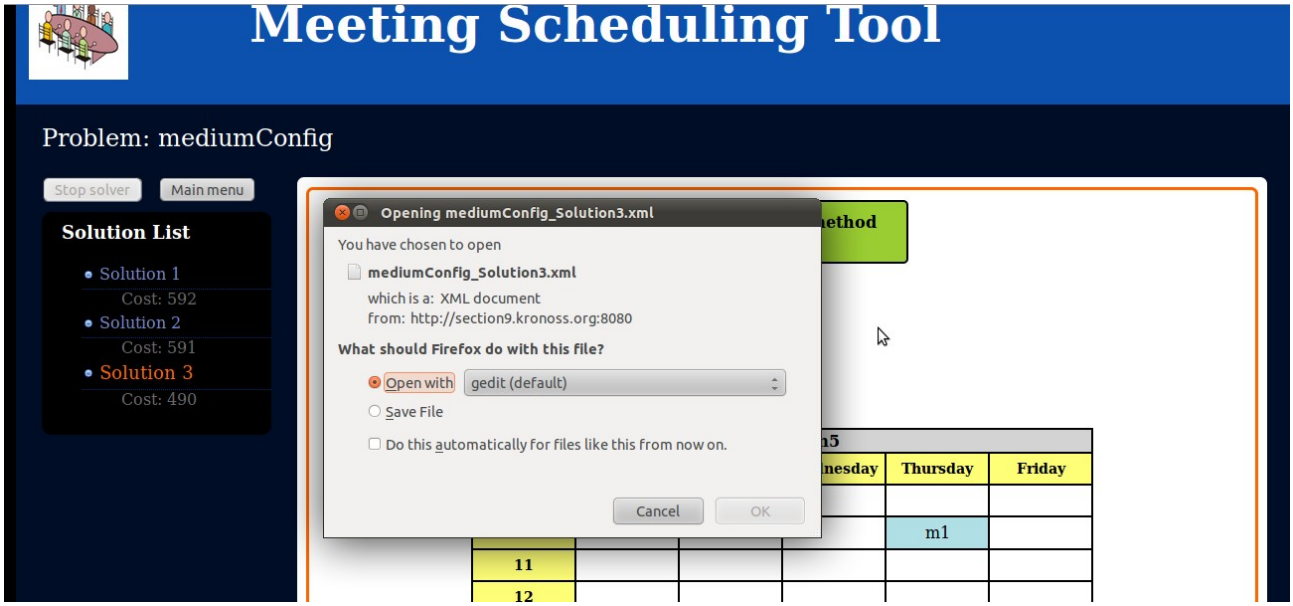

Figura 37 – Descargar solución

Anexo B - PMV Prediction Tool

El objetivo de la herramienta de predicción consistía en adaptar lo máximo posible las condiciones del edificio a las preferencias de sus usuarios, de forma que se aumentase el confort y se ahorrase energía al tener que modificar lo mínimo posible los sistemas de HVAC.

Para ello se utilizaron las extensas bases de datos de campo recogidas por la universidad de Macquarie[20]. Estos estudios contienen *feedback* de los usuarios basado en la escala *PMV ASHRAE*. Las bases de datos disponen de información acerca de la zona climática, *Id* de los usuarios, temperatura del aire, temperatura radiante, velocidad del aire y humedad. Mediante fórmulas sobre estas mediciones se obtiene el valor de la "sensación térmica" del usuario.

El sistema ITOBO almacena opiniones de los usuarios en el transcurso del tiempo. Estos datos están asociados a las lecturas de los sensores activos en ese momento, mediante la interfaz que se presenta en la siguiente sección del anexo. Se planteaba demostrar que, gracias a esta información, correctamente ponderada y agrupada se podría anticipar con mayor precisión las preferencias futuras de nuestros usuarios.

Para ello se desarrolló un algoritmo en lenguaje *Python*. La información mencionada se encontraba organizada en el motor de bases de datos *MySQL*. Recuperando secuencialmente esta información, nuestro programa creaba "distribuciones" de usuarios. Cada una de estas distribuciones correspondía a una "oficina" de usuarios que se encontraban en el mismo lugar.

El algoritmo se ejecutaba para cada una de las zonas climáticas. Parte de la base de datos se consideraba conocida, de tal forma que estaba disponible para la predicción, mientras que el resto se tomaba como condiciones futuras. De esta forma, era posible predecir los valores medios de la oficina, para más adelante calcular la tasa de error comparando con el *feedback* real recogido en los estudios de campo.

Para cada una de las situaciones futuras, se estudiaba la información relevante contenida en la parte conocida de la base de datos. Dependiendo de la relevancia de una información pasada, se le asignaba mayor o menor peso a la hora de predecir el nivel de confort actual para una "oficina" dada.

- Para considerarse relevante, la sensación térmica de la fila de datos conocida debía encontrarse dentro de un rango máximo de diferencia con la actual, a mayor similitud mayor peso.
- El *feedback* de los usuarios pertenecientes a la oficina que se está evaluando tiene más peso que los usuarios de otras oficinas.

20 http://aws.mq.edu.au/rp-884/ashrae_rp884.html

- Si la zona climática era la misma que la que se estaba evaluando, se le concedía más peso.
- La fiabilidad del usuario era tenida en cuenta. Si el usuario tenía *feedbacks* muy diferentes frente a condiciones similares se restaba peso.

El porcentaje de la base de datos conocido iba aumentando en sucesivas iteraciones, pudiendo demostrar así el beneficio de contar con un histórico de preferencias de los usuarios tal y como proponía nuestro proyecto.

Anexo C - User Feedback Interface

Como parte del proyecto ITOBO, se desarrolló también una interfaz para que el usuario pudiese dar su opinión sobre las condiciones de confort en el edificio. Esta interfaz está conectada a la base de datos *Oracle* que contiene:

- Información sobre los usuarios registrados en el sistema y su localización dentro del edificio.
- Lecturas de los sensores más cercanos al usuario.
- Información sobre los actuadores más próximos, que serán usados para regular las condiciones de acuerdo al *feedback* agregado de los usuarios de la oficina.

En primer lugar, el usuario tendrá que validarse contra la base de datos:

Figura 38 – Pantalla de acceso al sistema de confort en ITOBO

Una vez que el usuario halla sido correctamente validado, podrá ver su menú principal:

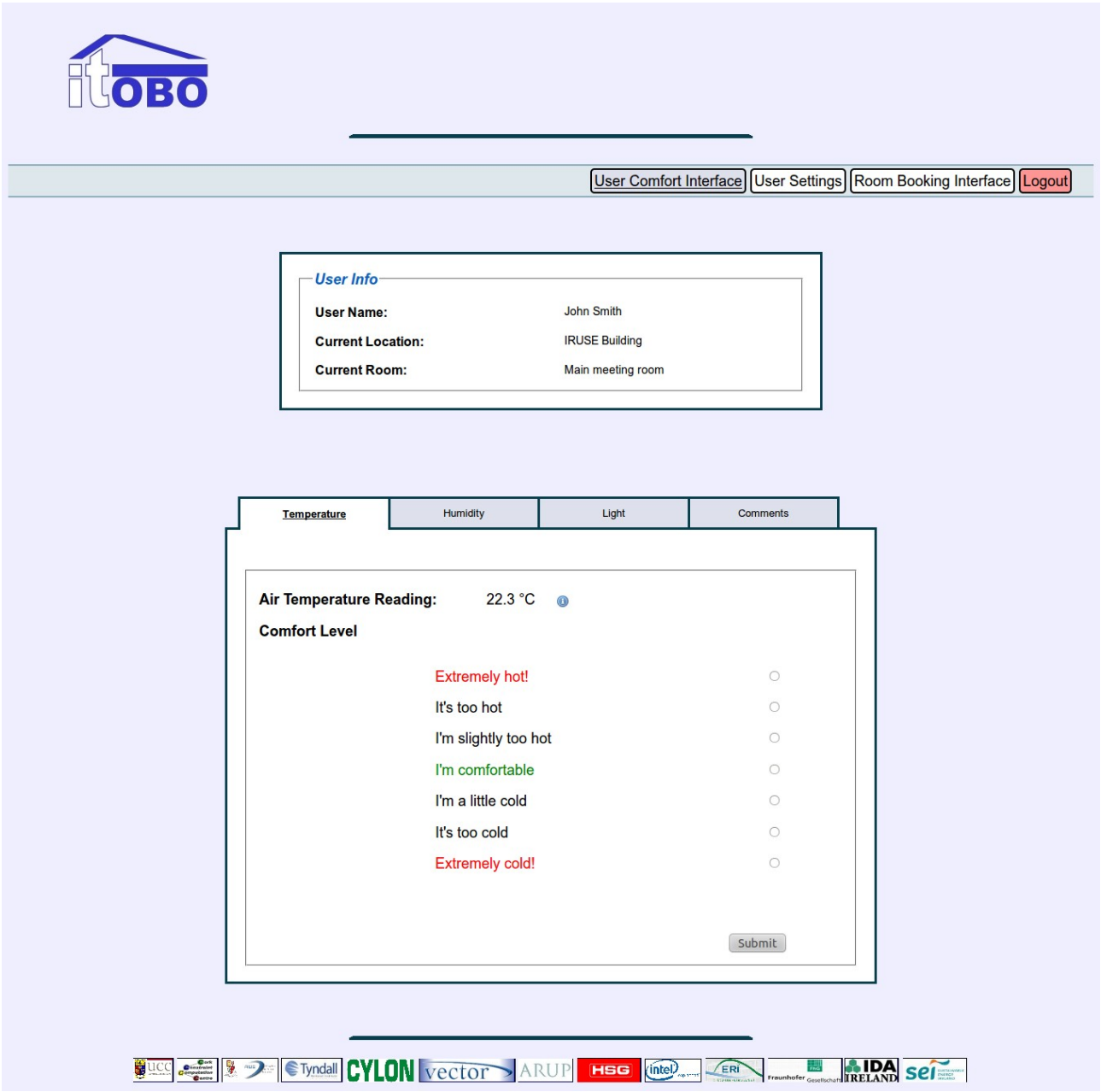

Figura 39 – Menú principal del usuario

En la tabla superior, el sistema ofrece información sobre la localización actual. En ciertas versiones del prototipo esta información cambiaba automáticamente de acuerdo a las lecturas de sensores RFID.

Podemos ver en el menú superior opciones para acceder a los datos completos del usuario y a la interfaz de gestión de reuniones.

En la parte inferior tenemos una tabla con diferentes pestañas donde el usuario podrá dar su opinión sobre las condiciones actuales.

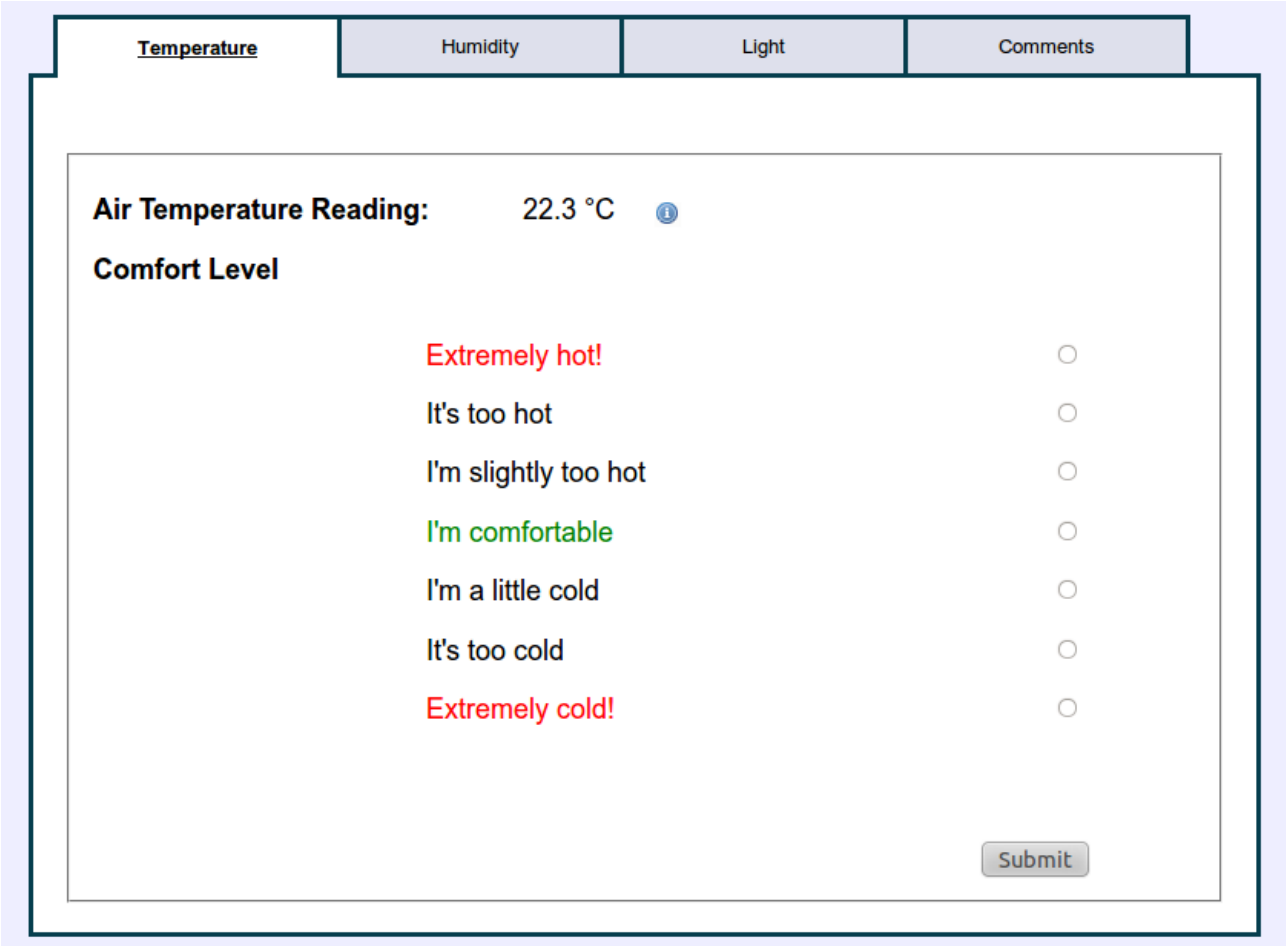

Figura 40 – Panel de temperatura

En el panel de temperatura el usuario puede ver la última lectura del sensor más próximo. Asimismo, puede enviar su opinión de acuerdo a la escala de 7 posiciones definida en el *PMV ASHRAE*.

Vemos un icono de información que aparecerá junto a las lecturas en todos los paneles de confort. Si situamos el ratón sobre el mismo, el sistema nos mostrará nuestro intervalo de preferencias de acuerdo a la información recogida hasta el momento.

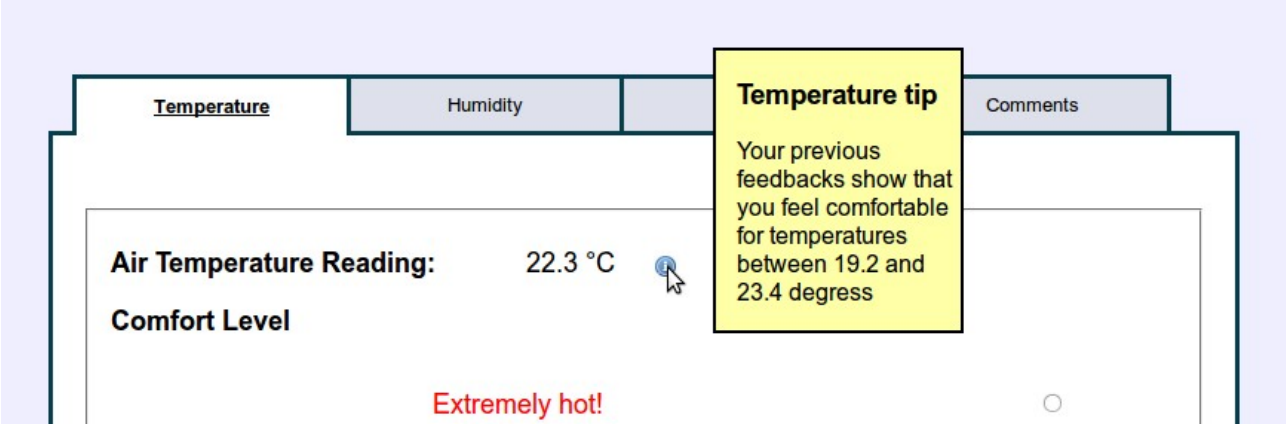

Figura 41 – Información sobre preferencias del usuario

Podemos observar las interfaces, similares en aspecto y comportamiento, para las lecturas de humedad y luminosidad del ambiente.

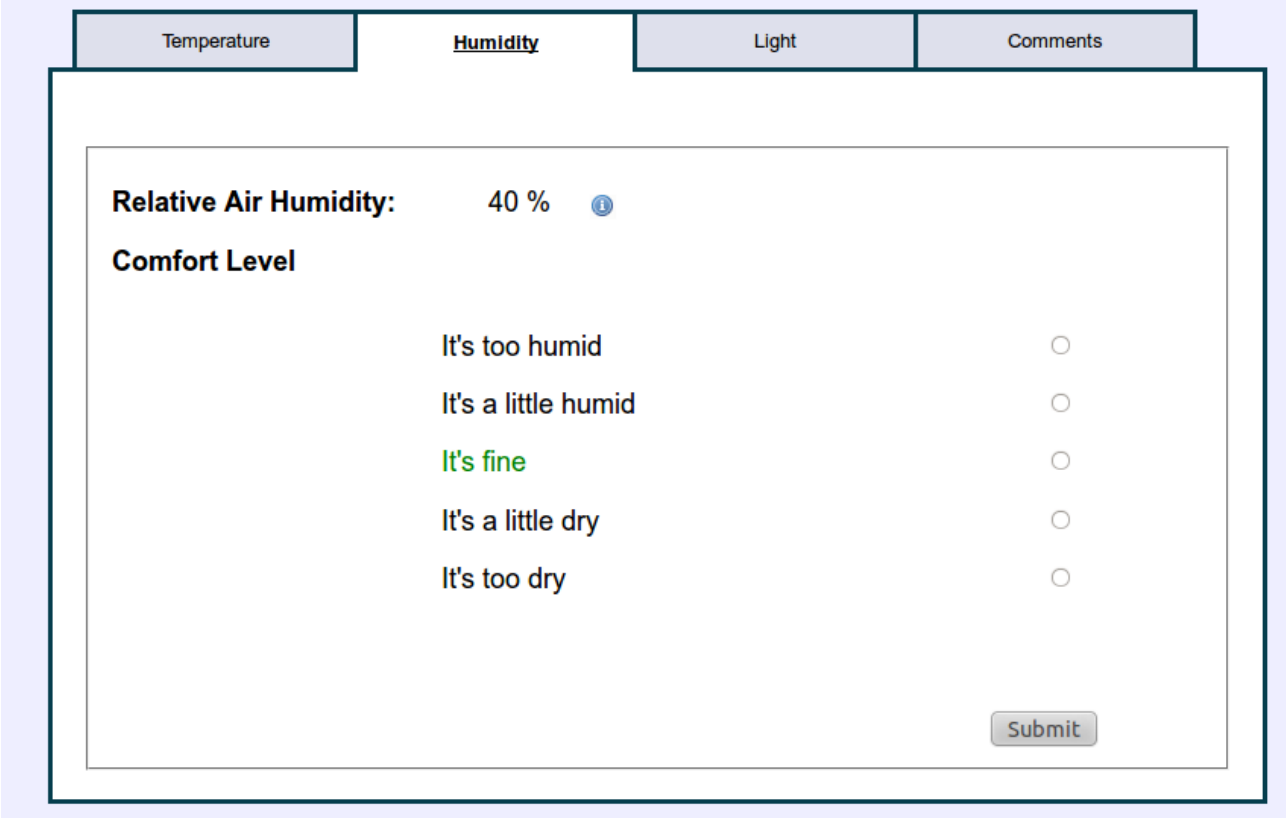

Figura 42 – Panel de humedad

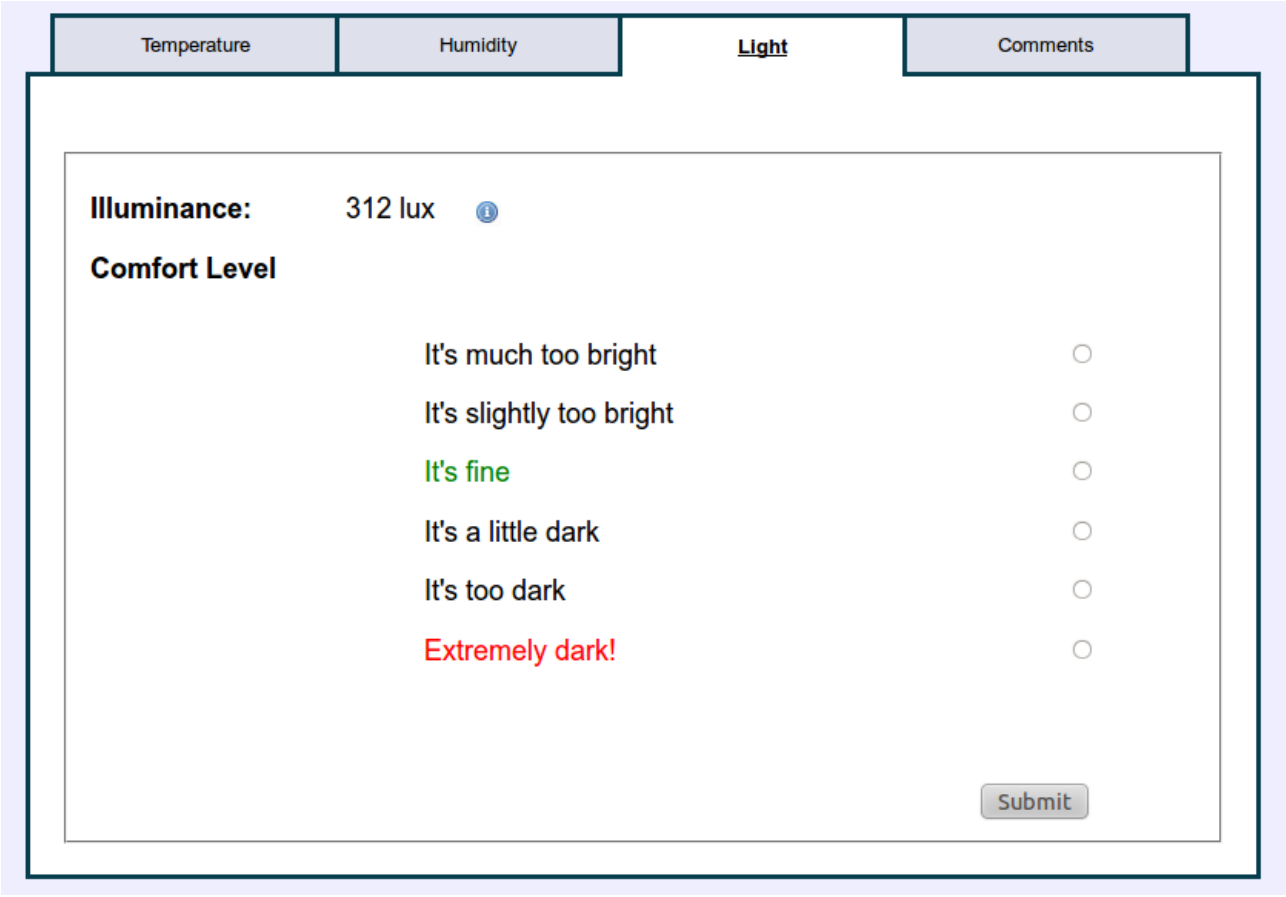

Figura 43 – Panel de luminosidad

El último de los paneles difiere ligeramente del resto, ya que fue ideado para cubrir los casos no contemplados en los que el usuario puede desear aportar su opinión.

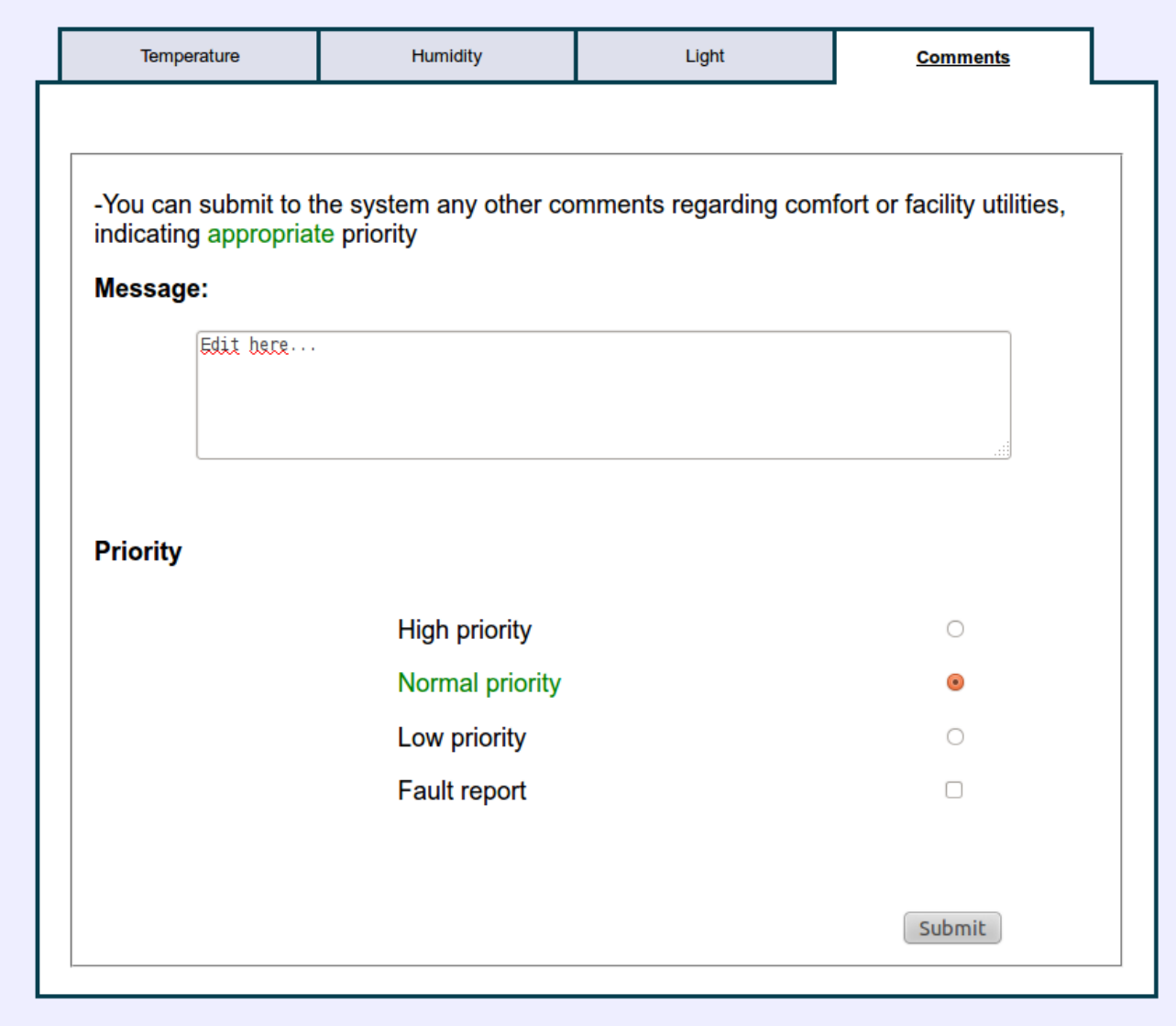

Figura 44 – Panel de comentarios

Mediante este panel podemos introducir cualquier texto, expresando la opinión sobre las condiciones de confort de nuestra habitación. Se invita al usuario a establecer la prioridad de su queja u opinión, indicando también si está comunicando un fallo en las instalaciones que requiera atención por parte del subsistema de mantenimiento.

Como se ha comentado, esta interfaz coloca marcas temporales a las opiniones de sus usuarios y las almacena en una tabla de la base de datos. Otros algoritmos recogían estos datos, siendo capaces de modificar los actuadores cercanos, – radiadores, ventiladores, accionadores de ventanas, humidificadores –, en tiempo real. En las sucesivas demostraciones realizadas con esta interfaz se experimentaba con el ajuste de las condiciones del edificio de manera dinámica para adaptarse a sus usuarios.

Glosario de términos y acrónimos

Ajax: *Asynchronous Javascript and XML.*

ASHRAE: Escala definida en el estándar ISO 7730, con el fin de recoger la sensación térmica de un individuo.

Constraint Programming: Paradigma de programación declarativo basado en la especificación de restricciones entre variables.

CSS: *Cascade Style Sheets.*

DTD: *Document Type Declaration.*

Genetic (Algoritmo): Familia de algoritmos de búsqueda local, basados en los conceptos y mecanismos básicos de la evolución natural.

Greedy (Algoritmo): Algoritmo que sigue la heurística de tomar la decisión local óptima para intentar alcanzar el óptimo global.

Hill-climbing: Técnica de optimización matemática perteneciente a la familia de la búsqueda local.

HTTP: *HiperText Transfer Protocol.*

HVAC: Subsistemas de confort del edificio. *Heat, Ventilation, Air Conditioning.*

ILS: *Iterated Local Search,* familia de algoritmos de búsqueda local.

ITOBO: *Information and Communication Technology for Sustainable and Optimized Building Operation.*

JSON: *Javascript Object Notation.*

MVC: Model-View-Controller, patrón de diseño de software.

NP-Hard: Tiempo polinomial no determinístico, clasificación de problemas en la teoría de la complejidad computacional.

PMV: *Predicted Mean Vote*. La respuesta media de un grupo de gente frente a una sensación térmica determinada, a partir de la escala ASHRAE.

Pool: Referente al algoritmo genético, grupo de especímenes que pertenecen a la misma generación de la búsqueda.

RFID: *Radio-Frequency Identification.*

RLS: *Repeated Local Search.*

SQL: *Structured Query* Language.

XML: *eXtensible Markup Language.*

XSD: *XML Schema.*

Bibliografía

- http://zuse.ucc.ie/itobo/ (página de presentación y documentación general del proyecto ITOBO)
- http://www.cis.cornell.edu/ics/compsust-org/crocs09/papers/brown-crocs09.pdf (Uso de la tecnología '*Constraint Programming'* en el contexto del proyecto ITOBO)
- Garey, M. R.; Johnson, D. S. (1979). Victor Klee. ed. Computers and Intractability: A Guide to the Theory of NP-Completeness
- http://aws.mq.edu.au/rp-884/ashrae_rp884_home.html (Bases de datos de confort de los usuarios, universidad de Macquarie)
- http://homepages.laas.fr/lopez/cours/CP/SEM-CSP.pdf (Introducción a la programación con restricciones)
- http://git-scm.com/documentation (Documentación de *git CVS*)
- http://www.springsource.org/documentation (Documentación entorno de programación MVC Spring para Java)
- Alan M. Davis: Operational Prototyping: A new Development Approach. IEEE Software, September 1992
- http://www.martinfowler.com/articles/continuousIntegration.html (Integración continua)
- http://sci2s.ugr.es/docencia/metaheuristicas/ILS.pdf (*Iterated Local Search*)
- http://www.obitko.com/tutorials/genetic-algorithms/ (Algoritmos genéticos)